Asymmetric Non-Uniform Proportional Share Scheduling

A thesis presented to
the faculty of
the Russ College of Engineering and Technology of Ohio University

In partial fulfillment
of the requirements for the degree
Master of Science

Michael S. Dunn
August 2010
© Michael S. Dunn. All Rights Reserved.
This thesis titled
Asymmetric Non-Uniform Proportional Share Scheduling

by
MICHAEL S. DUNN

has been approved for
the School of Electrical Engineering and Computer Science
and the Russ College of Engineering and Technology by

__________________________________________
Frank Drews
Assistant Professor of Electrical Engineering and Computer Science

__________________________________________
Dennis Irwin
Dean, Russ College of Engineering and Technology
Abstract

DUNN, MICHAEL S., M.S., August 2010, Computer Science

Asymmetric Non-Uniform Proportional Share Scheduling (197 pp.)

Director of Thesis: Frank Drews

Proportional ShareSchedulers allocate processing based on user defined weights such that each task receives a share of the processing time relative to its weight in any time interval. Proportional Share Schedulers minimize system jitter for time sensitive applications. To date the proportional share schedulers that support multiprocessors systems assume that the processors are symmetric. Newer processors support dynamic frequency scaling which saves power, but makes the system asymmetric.

We propose a load balancing technique that uses existing single processor Proportional Share Scheduler, to allocate fair shares of the system to tasks. The load balancer distributes the tasks in such away that each task receives a share equivalent to the share it would receive in a single processor system. However, this problem is complex and in general impossible to solve, so a pair-wise approach is used to simplify the problem.

Approved: 

Frank Drews

Assistant Professor of Electrical Engineering and Computer Science
# Table of Contents

Abstract ................................................................. 3

List of Figures ............................................................. 6

1 Introduction .............................................................. 7
   1.1 Importance of an Asymmetric NUMA Scheduler ................. 7
   1.2 Proposed Load Balancer .......................................... 7
   1.3 Contributions .................................................. 11

2 Background .............................................................. 13
   2.1 Schedulers ....................................................... 13
   2.2 Proportional Share Schedulers .................................. 15
      2.2.1 EEVDF ..................................................... 16
      2.2.2 SFQ and SFS ............................................... 17
      2.2.3 GR3 ....................................................... 20
   2.3 CFS ............................................................. 23
   2.4 Run Queue Models ............................................... 25
   2.5 Load Balancers .................................................. 29

3 Design ................................................................. 31
   3.1 The Problem ..................................................... 31
      3.1.1 Considerations ............................................ 35
      3.1.2 Method Overview ......................................... 36
   3.2 Algorithm ....................................................... 38
   3.3 Error Metrics .................................................... 46
      3.3.1 Defining the System ...................................... 46
      3.3.2 Performance Model ....................................... 53
      3.3.3 Infeasible Weight Assignments ......................... 60
      3.3.4 Criteria Implied by \( E \) .................................. 62
      3.3.5 Simplifying Values that Occur in the Criteria for \( E \) .. 65
      3.3.6 Simplifying the Criteria for \( E \) .......................... 68
         3.3.6.1 Underloaded/Overloaded: Symmetric ............... 75
         3.3.6.2 Underloaded/Overloaded: Overloaded is Faster ... 78
         3.3.6.3 Underloaded/Overloaded: Underloaded is Faster .. 82
         3.3.6.4 Both Overloaded .................................... 89
         3.3.6.5 Both Underloaded .................................... 90
      3.3.7 Optimal Choice of Weight for Metric \( E \) ................ 91
      3.3.8 Criteria Implied by \( R \) ................................ 99
      3.3.9 Simplifying the Criteria for \( R \) .......................... 107
         3.3.9.1 Underload/Overload .................................. 108
         3.3.9.2 Overload/Overload ................................. 113

...
List of Figures

3.1 Load balancer pseudo code, part 1 ........................................ 39
3.2 Load balancer pseudo code, part 2 ........................................ 40
3.3 The above graphs shows the weights where criteria 2a1 or 2a2 will be true. Processor k is under loaded by 1, and l is overload by a large amount. 66

4.1 Load balancer performance by error metric .......................... 132
4.2 CPU time lost to load balancing ......................................... 134
4.3 Load balancer performance by error metric .......................... 136
4.4 CPU time lost to load balancing ......................................... 137
4.5 Load balancer performance by error metrics ......................... 139
4.6 CPU time lost to load balancing ......................................... 139
4.7 Load balancer performance by error metrics ......................... 140
4.8 CPU time lost to load balancing ......................................... 141
1 Introduction

1.1 Importance of an Asymmetric NUMA Scheduler

Over the past few years the availability of computers with multiple processors has greatly increased due to the release of multiple core chips. Additionally, the availability of NUMA, non-uniform memory access, systems has increased with the decreasing price of commodity servers. With Quad-core chips and relatively inexpensive 4 socket NUMA mother boards, anyone who wants a high powered computer can easily obtain a 16 processor machine. Before the development of CPU frequency scaling, not only would each processor have had a constant operating frequency, each processor would have likely had the same frequency. In a modern machine those processors can reduce their speed in order to save power, additionally quad-core chips can change the frequency of each core individually. Together this means that today a commodity serve may have neither symmetric processors nor uniform memory access.

A proportional share scheduler allows the operating system to divide processing time amongst requesting tasks according to assigned shares with a high degree of accuracy. Current proportional share schedulers either assume that the system has only one processor [1–3], or that the system has only symmetric processors with uniform memory access [4, 5]. This paper proposes a load balancer that allocates tasks to processor such that allows each processor to use a single processor proportionally share scheduler, while each instance of the scheduler must only allocate the assigned processor.

1.2 Proposed Load Balancer

This paper proposes an Asymmetric Proportional Share load balancer, APS. APS aims to provide a scalable methodology to implement Proportional Share
Scheduling on a multiprocessing system. APS is designed to provide scalability over a large number of processors by using a distributed run queue model. In the distributed model each processor in the system has its own run-queue [6]. To ensure fairness within a run-queue, APS utilizes a single processor Proportional Share implementation. This implementation uses EEVDF, because of its good lag bounds [1], and the ease of adding state tracking to an EEVDF scheduler. EEVDF is defined in [1] and a summary is provide in section 2.2.1. APS attempts to distribute the total weight of the system in such away, that if the underlying PS scheduler is perfectly fair, the system as a whole will also be perfectly fair. In the case that such an assignment exists, finding the assignment equates to solving the subset-sum problem and is thus intractable [7]. If an assignment does not exist, APS will still need to assign each task to a processor. To find an approximate solution, APS seeks to minimize a given system wide error metric.

In order to balance the system each processor periodically runs APS. The processor currently running APS picks a processor that is more loaded than itself. If no processor is more loaded, APS will stop on this processor and try again later. To minimize lock contention, only two processors at a time are involved in task migration. If all of the processors were involved the lock contention would be the same as a centralized system, but without the fairness advantages. Bryant and Hawkes demonstrate the time wasted by lock contention on a 4 processor Linux 2.4 system in [8]. Linux 2.4 uses a centralized run-queue. After a pair of processors is established, the processor running APS and the more loaded processor, APS determines which of the processors should have weighted added to it. Let the processor gaining weight be the target processor and the processor losing weight be the source processor.
APS has three methods for reducing the error. Two methods are a direct minimization of the error metrics. The third is heuristic which may in some case make the error worse. The heuristic uses the selected pair to determine the error and will always lead APS to move weight to the less loaded processor. The minimization methods, however, consider the entire system. In some scenarios the error metrics are reduced by moving from the less loaded processor to the more loaded processor.

The next step for APS is to move weight from the source processor to the target processor. One method would be to calculate, from the error metric being used, the ideal amount of weight to move, and then use an SSP approximation algorithm to determine a solution. The SSP problem does not consider any costs in moving tasks from one processor to another. As a result it may move many tasks and only yield a small improvement in the balance of the system. While many heuristics could be used, APS selects from a set of rules the rule that yields the best improvement in error. If an estimate of the cost of moving tasks was available, that cost could be combined with the error result of a rule to discourage APS from selecting it.

1. Move a single task from the source to the target.

2. Move a single task from the source to the target and all tasks formerly on the target processor to the source processor. The target processor exchanges all of its tasks for one large task from the source.

3. Move a first fit set of task from the source processor to the target processor, and all tasks formerly on the target processor to the source processor. The first fit set is selected by assuming the target processor has no tasks and then adding each task on the source processor to the target if an only if adding it
will reduce the error of the target processor. Finally, each task formerly on
the target processor is moved to the source processor only if it will reduce
the error.

The rules were chosen to look for improvements in the case where the source
processor is overloaded and the target processor is underloaded. Unless the
system is balanced, at least one processor will be overloaded and at least one
processor will be underloaded. Since APS always looks for the most loaded
processor, when an underloaded processor runs APS it will seek an overloaded
processor to be the source.

Rule 1 looks for the single best task that can be moved to reduce the error. By
moving one tasks it is seeking the best error with the least cost. Rule 1 will not be
able to reduce the error when no task is sufficiently small, so rule 2 looks for a
single task large enough to replace the target processor’s current set of tasks. If
the source processor has such a large task rule 2 will try moving that task to the
target processor, and all of the other tasks to the source processor. When both
rules 1 and 2 can not find an improvement there are no tasks either sufficiently
small or sufficiently large. Rule 3 creates new empty set of tasks and then scans
through the tasks on the source processor, in the order that the run queue is
storing them. Each task on the source processor is added to the new set, if adding
to the task to the new set would reduce the error of the target processor, when the
new set replaces the set of tasks currently on the target processor. Next the new
set does not replace the set of tasks on the target processor, but is instead added to
the set of tasks on the target processor. Finally, each task that was on the target
processor prior to the new set being added, is moved to the source processor if
that move would improve the error of the system in its current state.
The idea is that rule, which only moves a single task, will be used most of the time. If the load balancer does use rule most of the time, it will help to reduce the cost of moving tasks. Rules and are designed to be used when the system is in a state where moving a single task will not help.

The move that would result in the least error is selected and committed by APS, assuming that a move with less error than the current state is found.

1.3 Contributions

This paper provides APS, an asymmetric proportional share load balancer that attempts to assign tasks to processors of varying performance in such away that each task will receive a fair share of the total system when each individual processor is scheduled using a single processor proportional share scheduler. A reference implementation of APS and an EEVDF scheduler are provided for the 2.6.21 Linux kernel. Starting with the 2.6 series the Linux kernel use multiple run queues to improve scalability of the scheduler with regard to the number of processors and tasks. This Proportional Share Scheduler is among the first to use multiple run queues to likewise improve scalability. CFS which was released in the 2.6.23 Linux kernel also supports multiple run queues.

APS is an asymmetric load balancer for Proportional Share Schedulers. APS is designed for a multiple run queue environment and distributes the tasks to each processor’s run-queue based on the current performance capabilities of each processor. In symmetric multiprocessor systems, SMP, all processors are assumed to have the same performance. APS is designed for an asymmetric system, thus it measures the performance of each processor and assigns a total weight to each processor in proportion to the processor’s performance. Additionally, the provided implementation of APS, uses information from the power management
subsystem to know which processors are currently running in low power mode and move tasks away from them as appropriate.

The load balancer provides a method for balancing Proportional Share tasks on NUMA systems. NUMA systems have multiple processor and multiple memory pools. Each processor is attached to one of the memory pools, and will lose performance when accessing any of the other memory pools. A basic multiple core system has only one memory pool, so each processor or core has equal access to the memory pool. Many servers though have multiple processors each with its own memory pool, these are NUMA systems. The load balancer only moves tasks when it finds that the move will improve the balance of system. Furthermore, the load balancer calculates the actual improvement of the move before committing to the move, making it easy to add a cost function to determine if the move is worth making.

The same balancing algorithm provided for balancing Proportional Share tasks on a single MP system can also be applied to moving tasks on a cluster of workstations each using Proportional Share Schedulers. The conservative nature of the load balancer in combination with the ease of adding a cost function make the load balancer well suited for moving tasks on a cluster, where moving a task is even more costly than it is on a NUMA system. Additionally the load balancer needs very little information about the system as a whole which is valuable to a cluster where getting information about the state of the cluster can be very expensive.
2 Background

2.1 Schedulers

The purpose of a scheduler within an operating system is to distribute the available processing resources amongst the tasks, which are requesting them. One of the goals common to most schedulers is to fully utilize the available processors. This means that a processor remains idle only if there are no more tasks ready for execution. A scheduler that achieves this goal is said to be work conserving [9, 10].

There are many schedulers with varying designs. Batch schedulers are only concerned with efficient use of processors and therefore schedule tasks to run until the task voluntarily leaves the processor. Desktop schedulers rapidly switch between tasks, so that no task in the system has to wait very long for its next time slice. This is particularly useful for tasks that spend most of their time waiting for input from devices such as the keyboard and mouse. Rapid switching however means the scheduler itself has to consume more of the processing time selecting tasks and performing context switches. Schedulers tuned for servers switch tasks frequently but not as frequently as the desktop schedulers.

Real-time schedulers are concerned with the amount of time allocated to each task. How this concern is handled varies amongst the different types of real-time schedulers. Hard real-time schedulers such EDF and RMA try to ensure that each task completes by a specified time [11]. Thus these schedulers are designed to ensure that each task is allocated enough time to complete its work prior to the deadline. Hard real-time schedulers guarantee that all tasks will complete by their specified deadlines, so long as the system engineer guarantees that system is not overloaded. Soft real-time schedulers are not as concerned with specific deadlines. Soft real-time schedulers attempt to allocate the available processor
time according to predefined shares of the systems processing time. This gives soft real-time schedulers more tolerance to system overloads, at the cost of hard guarantees of absolute allocated time.

Real-time schedulers have another concern often referred to as fairness. The specific definition of fairness varies amongst the different real-time schedulers, but the general idea is that each task is supposed to receive a particular amount of the available processing time over some period of actual time. Hard real-time schedulers are schedulers that must ensure tasks are completed by specified times referred to as deadlines. Hard real-time schedulers may waste processing time or other system resource to make the guarantee that a particular task will finish before it deadline. In general, real-time schedulers may give up being work conserving in exchange for fairness. Fairness in a hard real-time scheduler means ensuring that all tasks finish prior to their prescribed deadlines. Systems where failure is expensive enough to justify the cost typically use hard real-time schedulers.

Hard real-time systems themselves are quite expensive as many of the features of general purpose operating systems make task run times unpredictable. First for a hard real-time system to guarantee a task meets its deadline, it must know what the task’s deadline and how much processing a time a task will need to complete its work. Knowing the exact amount of time a task will need is difficult and is often done through statistical analysis. General-purpose operating systems can have interrupts occur at any time which will cause tasks to stop running for a long period. Real time systems use short interrupt handlers, which leaves much of the driver processing to the applications. Paging can also take a long amount of unpredicted time. Due to these long delays, real-time schedulers often do not have paging; meaning that the task implementers must handle all of
the complexities of memory management. Having to support these functions in the task greatly increases the complexity of the code and thus the cost of developing it.

In soft real-time schedulers, fairness more commonly refers to the amount of time a task has received. Soft real-time schedulers may have specific deadlines, but may also fail to meet the deadlines often by small amounts. Soft real-time schedulers try to divide the processing time in such away that all tasks receive an appropriate opportunity to complete their work in a timely fashion. Soft real time systems are designed to be tolerant of missed deadlines [11].

A hard real time system needs to ensure each task receives a specified amount of processing time over a given period. Reducing the amount of processing time allocate to other tasks, because a new task entered the system may cause a task already running to receive too little processing time and miss its deadline. One method to solve this problem is to specify a maximum limit on the systems workload at any given time, but most hard real time schedulers need to guarantee a task will complete within a certain period of time from when it started or some other event beyond the control of the scheduler. Proportional share schedulers are soft real time scheduler and in general do not guarantee completion by a hard set dead line. This allows proportional share schedulers to admit new tasks even though it may delay the existing tasks beyond their intended dead lines.

2.2 Proportional Share Schedulers

The scheduler proposed in this paper is a proportional share scheduler with load balancing. Proportional Share schedulers are soft real-time schedulers which define fairness by the weights of the tasks currently requesting processing time [2]. Each task in a Proportional Share system will ideally receive a proportion of
the total system time equal to its weight divided by the total weight of all tasks requesting processor time on any given interval.

Example 2.1. Assume a system has 3 tasks $t_1$, $t_2$, and $t_3$ and that these tasks have a weight of 3, 7, and 15 respectively. Then the system has a total weight of 25. Ideally on any interval in which all 3 tasks are requesting processing time, task $t_1$ will receive $\frac{3}{25}$ or 12% of the system’s processing time. Tasks $t_2$ and $t_3$ should receive 28% and 60% respectively.

The model proposed in [2] uses an ideal fluid flow system to define a proportional share scheduler. [2] defines the ideal fluid flow system as having one processor that can run all tasks simultaneously, each task receiving a certain percentage of the system. Thus each task is always granted the appropriate of $f_t(\tau) = \frac{w(t)}{S}$ at any time $\tau$, from equation (1) in [2], where $t$ is the given task, $w(t)$ is the weight of that task, and $S$ is the total weight of all tasks in the system at time $\tau$. The time $\tau$ is important, since tasks may enter or leave the system at any time. When ever a new task enters the system, all other tasks will receive slightly smaller amount of the portion of the processors time. Likewise, when a task leaves the system, all of the remaining tasks will receive slightly more processing time.

2.2.1 EEVDF

EEVDF, Earliest Eligible Virtual Deadline First, proposed in [1] and written by the same authors as the PS proposal in [2], is a Proportional Share Scheduler implemented on BSD, Berkley Software Distribution. As the name would imply EEVDF is similar to the hard real time scheduler EDF, Earliest Deadline First. Both EDF and EEVDF schedule that task with the nearest deadline first. EEVDF, however set deadlines in virtual time [1] which slows down as the system becomes more loaded.
Virtual time progresses at \( \frac{1}{S} \) the rate of actual time, where \( S \) is the total weight of all tasks on the processor \([1, 2]\). The link between virtual time and the total weight of each task, is how EEVDF meets all of its deadlines even though tasks may arrive at any time. Since virtual time progresses at a rate of \( \frac{1}{S} \), there are \( S \) actual time units per virtual time unit. EEVDF allocates to each task its weight in actual time-slices for each virtual time-slice. When a new task enters the system, the increase in the total weight slows virtual time such that the new task will also be able to get its share of time-slices.

Each task starts with a virtual eligible time equal to the virtual time at which it entered the system. The task’s starting virtual deadline is \( \frac{r}{w(t)} \) virtual time units after its virtual eligible time, where \( r \) is the actual length of a time slice and \( w(t) \) is the task’s weight. When the task completes a time slice its virtual eligible time becomes its former virtual deadline and its virtual deadline is moved forward another \( \frac{r}{w(t)} \) virtual time units. Since each task has a virtual deadline for each time slice it receives and no deadlines are missed, a task is off by no more than one time slice of actual time. EEVDF has an error of 1 time quanta, meaning that no task will have a lag greater than the chosen time quantum \([1, 2]\).

The implementation of EEVDF in \([1]\) and the implementation used in the paper have a scheduling run-time of \( O(n) \) on the number of tasks. The scheduler does not allow a task with a virtual eligible time in the future to run, therefore the scheduler must scan the list of tasks for the task with the least virtual deadline, which is also eligible.

2.2.2 SFQ and SFS

SFQ, Start-time Fair Queuing, proposed in \([3]\) is PS scheduler with similar properties as EEVDF. Like EEVDF, SFQ has constant error bounds on the achieved
fairness based on the maximum time quantum. Also like EEVDF, SFQ only runs on single processors systems. One significant difference between SFQ and EEVDF is that SFQ holds tasks in a priority queue, meaning that SFQ selects the next task to run in $O(\log n)$ time where $n$ is the number of tasks [1, 3]. SFQ also does not need to know the time quantum until a task has completed a scheduled run, thereby allowing the time quantum to be variable and easy handling of early task departures. Even though SFQ allows variable length time quanta, a singular long time quantum will increase the fairness error of the scheduler [3]. EEVDF handles early task departure in a fair manner, by simply adjusting the virtual time [1].

As the name would imply, the start-time of a task is the basis of SFQ. When a task initially starts, it is giving a start-tag equal to the current virtual time, which is zero if the system has just started. Whenever the system runs the scheduler, the scheduler selects the task with the least start-tag as the next task. When a task leaves the processor either voluntarily or by preemption its finish-tag is set to $F_t = S_t + \frac{q}{w(t)}$ where $q$ is the amount of time the task ran, $w(t)$ is the weight of the task, and $S_t$ is the start-tag of the task $t$. The tasks start-tag is then set to $S_t = MAX(F_t, v(\tau))$, where $v(\tau)$ is the current virtual time. SFQ defines the virtual time to be the start-tag of the currently running task or the greatest finish-tag tag assigned to any task if there is no currently running task.

SFS, Surplus Fair Scheduling, is a multiprocessor generalization of SFQ [4]. In the single processor case, SFS is simply SFQ. In the multiprocessor case, SFQ has no ability to handle the infeasible weight assignments along with a few other unfairness problems highlighted in [4]. SFS handles the infeasible weight assignments with a weight reassignment algorithm which adjusts the weight of overweighted tasks creating was is referred to as the closest feasible weight assignment [4]. This algorithm is an $O(n)$ algorithm, where $n$ is the number of
processors. Since only the \( p - 1 \) most weighted processor can be over-weighted or infeasible and SFS maintains a sorted doubly linked list of tasks only the last \( p - 1 \) tasks need to be checked [4]. SFS make the weight assignment feasible by simply reducing the over-weighted tasks to the maximum feasible weight, thus allocating them \( \frac{1}{p} \) of the system processing time where \( p \) is the number of processors.

Upon running the scheduler each processor in an SFS system will select the task with the minimum surplus instead of the minimum start-tag [4]. The surplus used by SFS is related to the start-tag and is calculated by \( \alpha_t = \phi_t (S_t - v) \), where \( \alpha_t \) is the surplus of task \( t \), \( \phi_t \) is the adjusted weight, \( S_t \) is the start-tag, and \( v \) is the virtual time. The start-tags and finish-tags for tasks are calculated using the same method as SFQ except that the adjusted weight \( \phi_t \) is used instead of the actual weight \( w(t) \). SFS calculates the virtual time in SFS in a similar manner as the virtual time in SFQ. Virtual time in SFQ is the start-tag of the currently running task, but multiprocessor systems have more than one running task, so SFS uses the minimum start-tag of any running task in the system. If there are no running tasks then SFQ uses the greatest finish-tag that it has assigned to any task, as the virtual time. SFS selects the virtual time identically to SFQ if there are no running tasks on any processor.

Virtual time in SFS is the minimum start-tag of any running task, and the greatest assigned finish-tag only if there are no running tasks. Whenever the virtual time is changed, SFS must recalculate the surplus, which could be every time any processor calls the schedule function. Recalculating the surplus takes \( O(n) \) time where \( n \) is the number of tasks. Additionally, when the surplus is recalculated the surplus queue must be resorted which takes \( O(n \log n) \) time [4].

SFS like SFQ maintains strong fairness amongst the running tasks. While SFQ is limited to running on a single processor, SFS can run tasks on multiple
symmetric processors. However, in SFS all processors must share a single run queue, meaning that while one processor is executing the scheduler, none of the other processors may access the run queue. Any processor that tries to schedule while another processor is already scheduling will continuously check the lock, performing no useful work, until the currently scheduling processor releases the run queue lock. The single run queue lock combined with a $O(n \log n)$ run time means that systems with more than a few processors are going to spend a lot of time waiting for to access the run queue.

### 2.2.3 GR$^3$

GR$^3$, Group Ratio Round-Robin, places all tasks into groups based on the weight of the task [5]. GR$^3$ then schedules the groups to run, and schedules the tasks within the groups, in a round-robin fashion. First each group has an order $k$ and the $k$-group contains all task with weights in the range $[2^k, 2^{k+1})$ [5]. Thus the number of groups is $O(\log n)$ where $n$ is the maximum weight of any task in the system. The groups then must be sorted by the total weight of the tasks in the group, $\phi_G$, such that the first group is the group with the greatest total weight of tasks. When a task departs or arrives the total weight of one of the groups will change and the sort order will have to be corrected, which letting $g$ be the number of groups takes $O(g)$ or $O(\log g)$ depending on what data structure is used. Since $g$ is often small and data structures with fast sort times can be difficult to implement in kernel space, using the more complicated structure may not be worth the small run time improvement [5]. Memory allocation is more likely to fail and requires more care in kernel space, making complex data structures more difficult to implement. In practice $g = \log n$ is likely to be small [5], particularly considering that $\log n$ can not exceed the size of the variable used to store the
weights. Each group also has a work counter that is incremented every time the group is scheduled. The scheduler uses work counter to determine if a group has gotten is fair share. If, for a pair of groups, the ratio of the work counters is greater than the ratio of the weights then the group in the numerator has gotten too much time. More precisely if \( \frac{W_{G_i} + 1}{W_{G_k} + 1} > \frac{\phi_{G_i}}{\phi_{G_k}} \) where \( W_{G_i} \) is the work counter of the \( i \)th group, group \( G_i \) has received more than its fair share compared to \( G_k \).

To schedule a task within a group, GR\(^3\) selects a task from the current group to be run and increments the work counter for that group. If the current group has received more than its fair share when compared to the next group i.e.

\[
\frac{W_{G_i} + 1}{W_{G_{i+1}} + 1} > \frac{\phi_{G_i}}{\phi_{G_{i+1}}},
\]

the next group becomes the current group. If the current group has not received its fair share then the first group becomes the current group. Finally, the selected task is scheduled. Since GR\(^3\) goes back to the first group whenever it does not go to the next group, the groups earlier in the array, the more weighted groups, get more time-slices.

Tasks in the \( k \)-group always have weight from \( 2^k \) to \( 2^{k+1} - 1 \). Each task has a relative weight within the group of \( \frac{w(t)}{\phi_{G_{min}}} \) where \( \phi_{G_{min}} \) is the minimum weight of any task in the \( k \)-group. This relative weight is always at least 1 and strictly less than 2. Every task also has a frontlog [5] that keeps track of how many time slices the system owes the task. Each time a task becomes the current task its frontlog is increases by it relative weight within the group. The scheduler then executes the current task for a time-slice and its frontlog is decremented by one. The current task will continuously get time-slices until it frontlog is less than one. After the task turn on the processor, GR\(^3\) selects the next group to have one of its tasks executed. Next time GR\(^3\) selects the \( k \)-group the next task will become the current task, thus the round-robin nature of GR\(^3\).
GR$^3$ has a $O(1)$ [5] run time, since it always executes the next selected group or task. GR$^3$ pays for its fast run time with a larger error than that of many other PS schedulers [1, 3, 4]. GR$^3$ has an error of $O(g^2)$ where $g$ is the number of groups [5]. While $g$ is small PS system can be quite sensitive to error and $g^2$ is not always so small. Consider a system with 5 groups and a 10 ms time-slice then that is roughly $5^2 \times 10\text{ms} = 250\text{ms}$ noting that $g^2 \neq O(g^2)$. 5 groups are only weights 1-31.

Like SFS, GR$^3$ can run on SMP systems [4, 5]. GR$^3$ uses a single run queue and weight reassignment similar to SFS. In GR$^3$, a processor can select a task that is already running. When a task that is already running is selected to run on a different processor it frontlog is increased as it normally would be, but the second processor runs the scheduler again to select a new task. This has the advantage that the currently running task will just keep running on the processor it is already, thereby making good use of the cache. GR$^3$, however can not run on asymmetric systems. Consider the example 2.2.

**Example 2.2.** Assume a system has 2 processors, A and B. Also assume processor A is twice as fast as processor B. Now assume the system has 2 tasks $t_1$ with weight 10 and $t_2$ with weight 5. The system will be completely fair if $t_1$ runs on A and task $t_2$ runs on B, so the weight assignment is feasible even though it does not meet the SMP definition of feasibility. Task $t_1$ will be in the 3-group and task $t_2$ will be in the 2-group. Now assume processor B gets to schedule first. Since the 3-group has more weight, it is selected first and its work counter is incremented. Processor B selects the only task, task $t_1$, to be executed. B increments the tasks frontlog by 1, then decrements it by one and finally schedules the task. Since \( \frac{1 + 1}{0 + 1} > \frac{10}{5} \) is false the first group is the new current group, but the first group is the 3-group. Now processor A gets to schedule. The 3-group is again the current group so it increases the frontlog on $t_1$ to 1 and increments the 3-groups work counter. Now $3 > 2$ so the 2-group is selected. Processor A, which still does not have a
task, increments the 2-group’s work counter and runs task $t_2$ setting its frontlog to 0. The work ratio is now $\frac{3}{2}$, so 3-group is the current group. The time-slice for task $t_1$ on processor B ends, but $t_1$ has a frontlog of 1 so the frontlog becomes 0 and B keeps running $t_1$. Now, A gets to schedule again and 3-group is current, so it will add 1 to $t_1$’s frontlog. 3-group will still be current, so A will have to increment $t_1$’s frontlog again, before selecting $t_2$ to run. To make the situation even worse, the numbers have not been adjusted for the fact that processor A is twice as fast as B. Processor A should be increasing and decreasing the frontlog by a factor of 2, to reflect that it is twice as fast.

2.3 CFS

CFS, Completely Fair Scheduler, is the new default Linux scheduler, which replaced the previous Linux scheduler in the 2.6.23 version. CFS, like the previous Linux scheduler has the primary goal of supporting interactivity. Additionally CFS tries to be fair in the same sense that PS schedulers are fair. The prior Linux scheduler gave a priority boost to tasks that spent most of their time sleeping, and longer time slices to tasks that spent most of their time utilizing the processor. CFS does not make such adjustments to tasks based on their sleep pattern, however tasks in the CFS scheduler, do accumulate processor shares while sleeping, up to an adjustable limit. In most PS schedulers, when a task goes to sleep the scheduler considers the task to have left the system and thus forfeits the remainder of it time slice. In CFS, a task that goes to sleep may retain its remaining time slice and build up extra time as if it was still waiting for the processor. CFS places a limit on this saved up time to prevent a task from sleeping for a long time and then monopolizing the processor.

CFS schedules tasks in a simple round robin fashion. The scheduler executes tasks in order of their virtual start-times and executes the tasks for a fixed amount
of virtual time. The virtual time quantum changes with the total weight and the number of tasks, but at any given time is the same for all tasks. In a stable state, the scheduler will operate as a simple Round Robin scheduler in virtual time. In actual time, however tasks are receiving variable length time slices. For each task virtual time progress inversely proportional to its weight, so a task with twice as much weight will receive twice as much actual time during its time slice. Thus, tasks with large weights will receive longer time-slices while tasks with small weights will receive short time slices, but all tasks receive the same number of time slices. Any tasks with a time slice less than 10µs will not be scheduled and will have to weight until it builds up 10µs of lag. The system does not keep a global virtual time, though it does track the minimum virtual time of any task in the system. Virtual time in CFS is mostly used order the tasks.

When a new task that has never ran before arrives in the system CFS inserts it as either the first task to run or the last task to run depending on the system configuration. If it inserted as the last task, the scheduler assigns the virtual start-time of the previous last task plus one, as the new task’s start-tag. If the scheduler inserts the task at the beginning, CFS assigns the same virtual time as the current minimum virtual time, to also be the start-tag of the new task. Since the scheduler breaks ties in FIFO order, CFS runs new tasks after all tasks that already have the current minimum virtual time.

CFS reinserts tasks that have previously left the system and are now returning based on their previous virtual start-time. The scheduler permits tasks that block or go to sleep to continue building a share of the processing time while the task sleeps. If the tasks prior virtual start-time is before the current minimum virtual start-time, the scheduler will move its virtual start-time forward to the current minimum upon waking up.
A task leaving the system either permanently or temporarily is simply removed from the queue.

Since CFS uses a red-black tree to store its tasks, therefore task departure, task arrival, and schedule are all $O(\log n)$ where $n$ is the number of tasks. While the runtime performance of CFS is comparable to SFQ the error bounds of SFQ are much better. SFQ has a maximum error of one time slice making the max error $O(1)$, the max error for CFS however is $o(n)$ where $n$ is the number of tasks. To demonstrate this consider example 2.3.

**Example 2.3.** Let the system be a single processor system running CFS. Assume the system has $n$ tasks which all start at the same time. Let the task $t_1$ have a weight of $w$, now let all the other tasks have a total weight of $w$. Since task $t_1$ has half of the weight, it should receive half of the time. All of the tasks entered the system at the same time, therefore their run order is arbitrary. Assume task $t_1$ runs last. Then when $t_1$ starts $\frac{1}{2}$ of the period has passed and in the ideal fluid flow system it should have receive half of that time i.e. $\frac{1}{4}$ of the period length. For a sufficiently large number of tasks, CFS determines the period length by a linear function on the number of tasks. Thus $\frac{1}{4}$ of the period is a linear function on the number of tasks and the error in this example is $\Theta(n)$. Therefore CFS has a maximum error at least $o(n)$.

### 2.4 Run Queue Models

There are three prominent run queue models for multiprocessor systems in research: centralized run queues, distributed run queues, and hierarchical run queues. Sivarama P. Dandamudi has done considerable research on run queue models in [6, 12, 13]. Systems that use a centralized run queue store all tasks in a single run queue and each time a processor needs a new task to the scheduler selects one from the run queue. At each selection, the scheduler is free to choose
whichever task is most in need of processing time from all tasks in the system. Thus, centralized run queues promote fairness within a system. This single run queue for an entire system comes at the cost of scalability. The run queue is a critical data structure that the system needs to protect by mutual exclusion. Furthermore, processors typically access the run queue from interrupt context, so if another processor is already holding the run queue lock a new requesting processor will have to spin and wait for the run queue. Contention for the single run queue makes the centralized run queue model too expensive for systems with a large number of processors.

Processor affinity is also a consideration when selecting a run queue model. Processor affinity is the tendency of tasks to perform more efficiently when the system schedules consecutive time slices on the same processor. Memory access is the largest factor for processor affinity. If a task is always running on the same processor there is a chance part of its memory will remain in the processor cache between runs. On uniform memory access systems, cache is the only memory consideration. In a NUMA system, the hardware physically links a set of processors to each memory bank. If a task runs on a processor that is not linked to the same memory bank that contains the task’s memory, the processor will have to retrieve task’s memory from the other memory bank. Fetching memory from a foreign memory bank requires usage of the global system bus, which can become a serious bottleneck for NUMA systems. The larger the system the more expensive it will become for task to move between processors. It will also become more expensive for a task to move as the amount of memory it uses increases. A centralized run queue gives no consideration to processor affinity.

Hierarchical run queue models use a tree of run queues with one centralized run queue at the root and each processor have its own run queue at the leaves. In
this model, each processor has a small queue of tasks waiting to for access to a processor. Each time the scheduler is called to select a task; the scheduler selects the task from the local run queue. When the local queue runs low on tasks it pulls a group of tasks from its parent queue. This continues up the tree until the root or central run queue is reached. New tasks enter the system at the root of the tree. This model solves the run queue contention problem by reducing the number of requests to the centralized run queue, since each access of the root queue will move several tasks down to one of its children. The hierarchical model makes no consideration of processor affinity. Each time a task runs, it may and probably will run on a different processor. In a batch scheduler like the one assumed by much of Dandamudi’s research, this is not a serious problem, since tasks will run to completion or at least for a long period of time. Proportional share systems schedule tasks using small time slices to reduce the amount of error. In systems with uniform memory access, tasks moving between processors may cause a small efficiency loss due to poor cache usage. In a NUMA system, moving a task from one processor to another may mean fetch memory from a distant memory bank or even moving a large block of RAM from one memory bank to another. This can make hierarchical run queues and small time slices an expensive combination for NUMA systems.

The distributed run queue model uses one run queue for each processor. Since each processor has it own run queue, there is very little contention for the run queue. Also, when the scheduler preempts a task, the task is placed back onto the run queue of the processor that just executed it, thus preserving processor affinity. While the distributed model has a low time cost even on large-scale systems, it has the considerable disadvantage of poor fairness. The processors in a distributed system may not have equal work loads. This imbalance will makes it
difficult to assign the correct amount of processing time to each task. This is a considerable disadvantage for a proportional share system, but the other two options are prohibitively expensive on large-scale systems. In a distributed system, the system assigns each task to a processor. If a processor has too many tasks or in the case of PS schedulers, too much total weight, then the tasks on that processor will lag behind. Similarly, if a processor does not have enough weight tasks on that processor will run ahead. Trying to assign a certain total weight to a processor from a given set of tasks is essentially the subset-sum problem which is NP-hard. Sleeping, blocking, swapping, completion, or new task arrivals may cause frequent changes to the task assignments, thus the system may need to adjust task assignments frequently. Distributed systems need a load balancer that can maintain a fair task assignment, while working within the constraints of processor affinity, limited run queue contention, and a fast performance.

APS attempts to maintain a fair weight distribution of proportional share tasks on an asymmetric multiprocessor system, while preserving processor affinity, contributing a minimal amount of run queue contention, and only using $O(p + n)$ run time.

A large number of multiprocessor real-time schedulers have been developed using the centralized run-queue model [4, 5, 14, 15], however this model suffers from scalability problems [16]. Centralized run-queues can be used to obtain near-optimal results for fair schedulers, but centralized run-queues cause serious performance problems on NUMA or distributed systems. A centralized run-queue on has places tasks on the next available processor with no regard for which processor has the best access to the tasks memory. Centralized run-queues also cause lock contention when many processors are trying to use the single queue [8].
2.5 Load Balancers

Load balancing is the process of partitioning a set of consumers across a set of producers. In the case of load balancing, the consumers are tasks and the producers are processors. Load balancers can be categorized into two types, static load balancers and dynamic load balancers [17, 18].

Static load balancers make load balancing decisions prior to run-time. Static load balancers use information about the tasks such as a performance profile to assign tasks to processors before the system ever starts to run. Each task is assigned a processor and will run on that processor every time is started and for the duration of the tasks execution time [17, 18]. Static balancing has the advantage that it allows for considerable amount of time to be spent searching for the best possible solution. The problem with static load balancing is that requires task with well known constant behavior [17]. A hard real-time scheduler such as EDF would likely be a better choice of scheduler for tasks with these qualities.

Dynamic load balancing places tasks at runtime. Task placement can take place at a variety of times in a dynamic load balancer [17, 18]. Some load balancers carefully place a task when it enters the system [17, 18], this can be when a new task is created or when it returns from blocking. Other load balancers may reassign tasks periodically while they are running [17, 18], and both carefully place a task when it enters the system and reassigns tasks as the system runs [17, 18]. The load balancer proposed in this paper only performs task reassignments, and does not make balancing decision when tasks are entering the system due to limitations of the operating system. Dynamic load balancers have the advantage of placing tasks according to the current state of the system in order to optimize the load when properties of the tasks are not known prior to run-time [17, 18]. The disadvantage of dynamic load balancing is that the time
spent determining the best allocation is time not spent performing the work the system is intended to perform.
3 Design

3.1 The Problem

A typical proportional share system tries to allocate processing time to tasks in proportion to their weights. On asymmetric systems, the performance of the processor must be taken into consideration. A task running on a high performance processor will complete more work than a similar task on a low performance processor, given the same amount of time. APS assigns tasks to processors, such that each processor will receive processing performance in proportion to it weight. The performance is empirically measured, and each processor is considered to produce a number of performance units per unit time. The provided implementation uses the bogoMIPS which is calculated by the Linux kernel. An ideal method for determining the performance of a processor is beyond the scope of this paper.

The load balancer considers each processor to have a load of the ratio of the total weight of the processor’s tasks to the processor’s performance. The load balancer then relies on each processor to have an instance of a single processor proportional share scheduler to ensure that the assigned tasks get the appropriate share of that processor’s time. Since each processor is considered to produce a certain amount of performance units per unit time, the amount of time allocated by the single processor scheduler is directly related to the amount performance received by the task.

As a decision problem the system can be described as follows. Given a set of weights $N \subset \mathbb{N}$, and a set of sums $S$ subset of the natural numbers, containing $\lambda$ sums. Determine if $N$ can be partitioned into $\lambda$ subsets, $T_i$ such that $\sum_{t \in T_i} \frac{t}{S_i} = \sum_{t \in T_j} \frac{t}{S_j}$ for all $j$. If the sum of the elements of $S$ is equal to the sum of the
elements of $N$, then this problem is as hard as the subset-sum problem. The sum of $S$ and the sum of $N$ can be made equal by applying a scalar $c$ to all elements of $S$ without changing the problem since, $\sum_{t \in T_i} \frac{1}{cS_i} = \sum_{t \in T_j} \frac{1}{cS_j}$ if and only if $\sum_{t \in T_i} \frac{1}{S_i} = \sum_{t \in T_j} \frac{1}{S_j}$. The subset-sum problem has one sum $s$ and seeks one subset of $N$, in contrast this problem has several sums, seeks a subset for each sum, and must use all elements of $N$.

To transform the decision problem into an optimization problem first a function $F(S, N, Q)$ is defined, where $Q$ is a set of subsets $T_i$ representing a possible solution to the decision problem. Furthermore, $F(S, N, Q) = 0$ must be true if and only if $Q$ is a solution to the decision problem, in all other cases $F(S, N, Q) > 0$ must be true. The optimization problem is then consists of finding a set of subsets $Q$ that minimizes $F$ for the a given set of sums $S$ and set of weights $N$. Two possible function for $F$ are presented and studied in this paper, metric $E$ defined in 3.3.1 and analyzed in 3.3.4, 3.3.5, and 3.3.6 and metric $R$ defined in 3.3.1 and analyzed in 3.3.8 and 3.3.9.

The problem of balancing the load is a generalization of the Subset-Sum Problem, which is NP-complete. The definition of subset-sum decision problem is as follows: Given a set of positive integers, $N$, and an integer $s$ determine if there exists a subset of $N$, such that the sum of its elements is equal to $s$. In many cases, a subset is need whether or not an ideal subset exists, that is a subset that meets the criteria of the decision problem. As a result, the subset-sum problem is also well studied as an optimization problem, which finds a solution close to the ideal solution. The definition of the subset-sum optimization problem is as follows: Given a set of positive Integers, $N$, and integer $s$, find the subset with the largest sum, such that the sum is less than or equal to $s$. There are many known approximation algorithms for this problem [19–22].
To generalize the subset-sum optimization problem into the load balancing problem, consider a set of integers $S$, such that the sum of the elements in $S$ is equal to the sum of the elements in $N$. Each element of $S$ represents a target sum, $s$, in the subset-sum problem or the performance of a processor in the load balancing problem. Each element of $N$ represents an integer from the set of choices in the subset-sum problem or the weight of a task in the load balancing problem. The elements of $S$ are assumed to be scaled such that the total of $S$ is equal to the total of $N$. Thus each element of $S$ can be considered to be a subset-sum problem with the additional constraints that each element of $N$ can only be used in one solution sets and each element must be used in one of the solutions sets. These constraints are necessary, since each task requires a processor and no processors may share a task. The subset-sum approximation problem requires that the solution does not exceed the target sum, while the load balancer will look for the closest solution to the target. This is necessary since, a solution less than the target means that a processor is underloaded, and an underloaded processor implies that there is also an overloaded processor. If any of the solutions do not find a subset totaling to the target sum, then one of the other processors must exceed that sum to place all of the tasks. Essentially the load balancer must approximate many subset-sum problems which all draw from a common set of integers, $N$.

**Example 3.1.** Consider the following example of the subset-sum problem with multiple disjoint subsets. Let $N = \{ 1, 2, 3, 4, 5, 6, 7, 9, 16 \}$ Let $S = \{ 10, 11, 15, 17 \}$ In this case 4 sets of integers from $N$ must be found each totaling a distinct element from $S$. $N1 = \{ 5, 3, 2 \}$ $N2 = \{ 7, 4 \}$ $N3 = \{ 6, 9 \}$ $N4 = \{ 1, 16 \}$

In this example $N$ represents the weights of the set of all tasks in the system. Each element of $S$ represents the performance of a processor. $N1$, $N2$, $N3$, and $N4$ would be the sets of tasks placed onto the processor by the load balancer.
Balancing the load of a system is an online problem. The load is the ratio of the weight of the tasks and the performance of the processors, thus the load changes any time the tasks or processors change. Tasks enter and leave a system constantly. Any time a task has to wait for disk access, network bandwidth, or even swapping it will leave the system and reenter when the resource is ready. Many proportional share systems consider processors static, but the performance processors can also change. A system with power management may frequently change the voltage to a processor, thus altering its performance. Some midrange systems and mainframes are able to add and remove processors while the system is running to compensate for additional short-term load. While there are systems and tasks that face none of these considerations, there are many systems where the run-time environment is dynamic and require frequent load balancing. For these reasons the solution proposed in this paper is an online solution designed to run periodically to update the solution as the state of the system changes.

The load balancer faces an additional problem that is not considered by the subset-sum problem, the cost of task migration. When the load balancer is given a set of tasks and processors to balance, it is also given a start state. The start state is the current assignment of tasks to processors. Changing this state requires moving tasks, which time consuming and uses critical resources. The load balancer finds a better solution by considering moving tasks from this starting state.

The cost of moving a task varies by the size task’s working set, the system, and the specific processors involved. Moving a task between to core on the same physical CPU has a low cost of acquiring two run-queue locks and some cache misses. The cost of moving a task to a different physical CPU that share the same memory pool will add additional cache misses, but it still low. Task migration though can become expensive when moving a task between processors that do
not share memory, as is the case in NUMA systems. When the task moves to a processor using a different memory pool the new processor will have to access the tasks memory over the interconnect bus, or the operating system will have to move the tasks working set to the new memory pool. Some clusters work similarly to large NUMA systems, by a network between machines that acts like a memory interconnect bus between processors. The load balancer will move the tasks for a small benefit even if the cost is high. On small multiprocessors system the cost should be small, but on large scale system this cost should be considered.

A cost function can be used to estimate the cost of moving the tasks between processor and then compared to the benefit of the improved balancer. A good cost function would be highly dependent on the system being used and is left to further research. In this paper the cost is always assumed to be zero.

3.1.1 Considerations

This paper proposes a load balancer and a new asymmetric model for Proportional Share Scheduling. To support the new model and load balancer two error metrics are also introduced. A theoretical analysis of the error metrics is provided to determine when this or any load balancer will improve the system and in what scenarios it would make the system error worse. However, no analysis of how these error metrics actually apply to quality of service of a real-time task is provided in this paper.

In order to simplify the problem, the proposed load balancer does not consider that cache or memory effects of moving tasks. When a task moves from one processor to another, the system will have to move its cache new the new processor. Additionally, if the system is a NUMA system, the working set will also need to be accessed by the new processor. This may mean that the memory of
task will have to be moved to the new processor local memory set, or that the new processor will have to access the task’s memory in a foreign memory set. The time lost to these problems is not considered by this implementation.

The proposed load balancer depends on a method of measuring the performance of a processor, but this paper provides no analysis of what method should be used. It is anticipated that the nature of the actually tasks being balanced would determine the best method for measuring the performance of a processor. The provided implementation uses the Linux kernel’s built-in method of BogoMIPS to measure the processors performance, but no analysis was done to determine the usefulness of the measure.

3.1.2 Method Overview

The timer interrupt is triggered after a specified amount of time by a clock in the system. Each processor receives it own timer interrupt. The Linux kernel uses this timer to execute the scheduler, to implement sleep function, and to execute the load balancer. APS is executed from this interrupt handler every time the normal Linux load balancer is executed. Each time the interrupt is trigger the Linux kernel determines the next time the interrupt handler should run and set the timer. Additionally, the load balancer is not executed every time. If the load balancer doe not move any tasks the kernel will wait longer before executing again; if the load balancer does move tasks the kernel will wait less time before executing it again. The result is that the load balancer, is executed a couple of times a second.

When the load balancer is executed, it first picks a pair of processors that it will try to balance. The first processor is the processor that the load balancer is running on, and the other processor is the most loaded processor that available
for balancing considering the total weight of the tasks on the processor and its current performance. If the load balancer recently picked this pair of processors and was unable to make an improvement, that processor will not be considered when selecting the most loaded processor. By picking a pair of processors, the load balancer never holds the run queue lock for more than two processors at the same time.

Once the load balancer has selected a pair of processors, the load balancer tries to improve the balance of the system by minimizing the error metrics discussed in 3.3. The load balancer does this by considering a variety of possible moves and then selects the move, which minimizes the current error. The load balancer must know the total performance of the system, the total weight of all tasks in the system, the total weight of the tasks on both processors in the pair, and the performance of both processors in the pair, in order to calculate the improvement in the error metric. When the load balancer is considering moves, it will consider moving individual tasks, sets of tasks, and exchanging tasks between the two processors. Tasks may be moved to either processor, it some case it is even advantageous to increase the load of the most overloaded processor. This can happen when both processors in the pair are overloaded and the more overloaded processor has higher performance than the other processor.

**Example 3.2.** Consider a system with 2 processors k and l. Assume k has a performance of 100 and tasks with weights 100 and 21 for a total weight of 121. Now Assume that l has a performance of 25 and tasks with weight 25 and 5. Also, assume that the total performance of the system and the total weight of the tasks are equal. In this scenario processor k is 21% overloaded 121/100, and processor l is 20% overloaded 30/25. The system is then has a total overload of 41% plus the total for the rest of the system. If the weight 5 task is moved from processor, l to processor k, then processor k will have a total
weight of 126 and processor l will have a total weight of 25. This yields overloads of 26% for processor k and 0% for processor l. The system now has a total overload of 26% plus the rest of the system.

The load balancer then compares the best move it could find to the current state. If the move will improve upon the current state then the load balancer moves all of the required tasks. If the currently running task is one the tasks to be moved then it is marked to be moved later. Since the load balancer finds the best state by calculating the error metric for each state, the load balancer will never increase the error.

The behavior of the load balancer is not deterministic. When the load balance selects a pair of processors it is actually only selecting one processor, the other is always the processor that is executing the load balancer. Thus, the order in which processors run the load balancer determines how the load balancer selects the pairs. The kernel runs the load balancer from inside the context of the timer interrupt. Which processor receives and responds to the timer interrupt first is a race condition. Additionally, any time an interrupt handler uses a resource it locks it with a spinlock, which is uninterruptable. Tasks may also hold locks that the scheduler needs, and delay its execution. Thus altering the order in which the processors run the load balancer. Because so many uncontrolled factors influence when the load balancer runs on which processors the behavior of the load balancer may be different on different runs even when the start states are identical.

3.2 Algorithm

In the Linux kernel the load balancer is executed from a soft-IRQ, which is a software emulated interrupt. The Linux kernel periodically fires the soft-IRQ
pss_load_balancer() {

    processor current, heavy
    current = cpu(smp_processor_id())
    heavy = find_heaviest_cpu()

    if (heavy = NULL)
        clear_flag(no_improvement, all_processors, current)
    return

double_lock(current, heavy)

    if (W(current) * p(heavy) < W(heavy) * p(current))
        goto out

current_error = error(current, heavy)

    error[move_single] = error_if_move_single(current, heavy)
    error[exchange] = error_if_exchange(current, heavy)
    error[firstfit] = error_if_firstfit(current, heavy)

    best_error = MIN(current_error, error[*]);

    Figure 3.1: Load balancer pseudo code, part 1

SCHED_SOFTIRQ from the timer-tick’s interrupt handler. This soft-IRQ then in
turn runs the Linux kernel’s normal load balancer routine. The PS load balancer is
if (best_error = current_error)

    set_flag(no_improvement, heavy, current)

else if (best_error = error[move_single])
    do_move_single();

else if (best_error = error[exchange])
    do_exchange();

else if (best_error = error[firstfit])
    do_firstfit();

if (ready_tasks(heavy) = 1 and current_waiting_to_move(heavy))
    wakeup(heavy->migration_thread)

out:

double_unlock(current, heavy)

Figure 3.2: Load balancer pseudo code, part 2

likewise called from the soft-IRQ just prior to the time-sharing load balancer provided by the Linux kernel. Since every processor has its own timer tick, every processor will periodically run the load balancer.

The first thing the load balancer does is find the most loaded processor in the system. The system calculates the load of a processor from the total weight of all
the tasks on a processor and the performance of the processor. Let $W(T)$ be the total weight of processor $n$ and $p(n)$ be the performance of processor $n$. To determine the performance, the load balancer simply uses the Linux measure of BogoMIPS. BogoMIPS or bogus MIPS, uses a loop to measure the number of instruction per second the processor can execute. Developing a better performance function is beyond the scope of this paper. The load of processor $k$ is greater than the load of processor $l$ if $W(Q(k)) \times p(l) > W(Q(l)) \times p(k)$. By comparing the cross product instead of the ratio $W(Q(k))/p(k)$, integer division is avoided which is particularly useful since the ratio is often less than one. By policy, the float point arithmetic and floating point registers are not allowed inside the Linux kernel [23].

APS does not consider certain processors when selecting the most loaded processor. A processor will not be selected as the most loaded processor if any of the following are true:

- The candidate processor is less loaded than the current processor $[W(\text{current}) \times p(\text{candidate}) \geq W(\text{candidate}) \times p(\text{current})]$.
- The candidate processor is currently waiting on the current task to move to another processor.
- APS could not make an improvement in a previous attempt to balance against the candidate and the candidate has acquired no new tasks.

Since the currently running processor will never select a processor less loaded then itself the most loaded processor will never perform the work of running the load balancer. By doing this, less loaded processors assume most of the responsibility for running the load balancer.
Sometimes APS will determine that the best move for balancing the system involves moving a task that is currently running on a processor. APS however cannot move a running task to another processor while it is still running. In this situation, APS flags the task with the processor that should acquire the task next. The scheduler will then move the task when its current time slice expires; by queuing the task on the next processor, instead of the processor it just finished utilizing. When the current task on a processor has such a move pending, other processors should wait until the move scheduler has completed the move, before trying to balance against that processor.

Whenever APS fails to make an improvement for a pair of processors, the running processor flags the other to indicate that APS could not improve the balance. Next time the same processor runs the load balancer, it will not try to balance against a processor on which it previously set this flag. APS will clear the flag if the running processor cannot find any processors to balance against or the processor for which the flag is set gains an additional task. If APS cannot find a more loaded processor to try to balance against, the load balancer simply exits.

Next, the load balancer acquires the run-queue locks for the current processor and the most loaded processor. After acquiring the locks, the load balancer checks again to ensure that the most loaded processor is still more loaded than the current processor. If it is not, the load balancer will exit and wait to run again.

Once APS has select a pair of processors it attempts to reduce the error for the system, rearranging the tasks on the the selected processors. APS has multiple methods for determining the error, which are discussed in 3.3. The error metric will determine the task arrangements that APS will favor. In many cases the error is improved by moving weight from the more loaded processor to the less loaded processor. In some cases the error is reduced by moving weight from the less
loaded processor to the more loaded processor. The cases in which this happens are examined in the discussion of the error metrics for which they happen. At this point APS decides which processor should gain more task weight, this is the target processor. The processor loosing task weight is the source processor.

APS now determines the error values that can be achieved by moving weight from the source processor to the target processor. The load balancer starts by determining the error contribution of the selected pair in their current state. The load balancer then determines what the error will be after each move it can make. APS can make the following moves:

1. Move a single task from the source to the target.

2. Move a single task from the source to the target and all tasks formerly on the target processor to the source processor. The target processor exchanges all of its tasks for one large task from the source.

3. Move a first fit set of task from the source processor to the target processor, and all tasks formerly on the target processor to the source processor. The first fit set is selected by assuming the target processor has no tasks and then adding each task on the source processor to the target if an only if adding it will reduce the error of the target processor. Finally, each task formerly on the target processor is moved to the source processor only if it will reduce the error.

When considering moving a single task the load balancer looks for the task that will reduce the error the most when moved to the target processor. To do this, it simply checks what the resulting error would be for moving each task that is on the source processor. APS selects the task that reduces the error for the pair by the
largest amount, even if it would overload or otherwise worsen the error on the target processor.

For the second option, the load balancer looks for a task on the source processor that is close to the ideal amount of weight for the target processor. If APS finds one, it will move that task to the target processor, and move all of the tasks that were previously on the target processor to the source processor. APS is trying to increase the total weight of the target processor, thus it only uses this move when it finds a task on the source processor that has more weight than the total weight of the current processor.

In the third move, APS uses a first fit method to select a new set of tasks for the target processor from the set of tasks on the source processor. For this move, APS assumes that it will be moving all of the weight on the target processor to the source processor, thus the target processor has a total weight of zero. It then walks the queue of tasks on the source processor and adds each one to the target processor if adding to the set will reduce the error of the pair. Then APS walks through all the tasks that were on the target processor at the start of the move, and it moves each of those tasks to the more loaded processor if it will reduce the error. Some tasks that the load balancer assumed it would be moving away from the current processor may stay on the current processor. The order of tasks in the queue is arbitrary, thus the results of this method are not deterministic.

The last move option only applies when there is an infeasible task on one of the processors in the pair. APS find the most weighted, infeasible task in the pair, and moves it to the faster processor. APS then moves all other tasks to the slower processor. If there are only two processors in the system and there is an infeasible task this move is likely to produce the optimal solution. If however, there are more than two processors in the system, then the total weight of all tasks in the
pair other than the infeasible task may be greater than the weight of the infeasible task. Therefore, when there are many processors the other moves may find better solutions, even if infeasible weights are present.

The load balancer evaluates the error that would be achieved by each move and compares the results. If a future implementation estimated the cost of a move, this is where it would be done. Some systems incur a significant cost to move a task, and the benefit to the error may not be worth the cost of moving the tasks. The benefit of fixing a small error in a system with a rapidly changing set of tasks is small, but the cost of moving tasks to reduce the error in a large scale system such as a NUMA system or cluster may be very high. Small MP systems with uniform memory access have almost no cost to moving; therefore, this implementation does not estimate the cost of moving the tasks.

Now that the load balancer has gathered all the information it needs, it commits the move with the lowest error value. If the current state has the lowest error value, then the load balancer makes no move and sets the no improvement flag on the more loaded processor.

If the currently running task on the more loaded processor is the only task and it is flagged to be moved, the load balancer will wake up the migration thread. If there is only one task on a processor then its time slice will never expire, and the scheduler will never actually move the task. The migration thread is a special kernel thread that always exists, is always ready to run, and has a special priority higher than any other task in the system. Waking up this task ensures that the currently running task will be preempted in the near future and therefore, moved to its new processor. The normal function of the migration thread is to move tasks away from a processor that need to be moved immediately, such as moving all the tasks off a hot plug processor that is being removed. When the migration thread
wakes up it will see that its queue empty, the migration task will run the normal
Linux load balancer in process context and go back to sleep. Since the last PS task
was just moved away from the processor, if the load balancer failed to find any
tasks for this processor and it has no time sharing tasks, the processor will run the
load balancer one more time before idling. The load balancer will continue trying
to find tasks for the processor each time it is triggered by timer interrupt.
Finally, the run queue locks are released and the load balancer exits.

3.3 Error Metrics

3.3.1 Defining the System

In this paper, two primary error metrics are used, maximum absolute relative
error and total absolute relative error. The maximum absolute relative error,
denoted by $R$ is the greatest value of $|r_n(Q(n))|$ for any processor in the system.
The total absolute relative error, denoted by $E$ is the sum of the absolute relative
error, $|r_n(Q(n))|$, for each processor, $n$, in the system.

$$N \quad \text{is the set of all processors in the system.} \quad (3.1)$$

$$n \quad \text{is used to denote a individual processor.} \quad (3.2)$$

$$T \quad \text{is used to denote a set of tasks.} \quad (3.3)$$

$$t \quad \text{is used to denote an individual task.} \quad (3.4)$$

$N$ is the set of all processors in the system. It is assumed that $N$ is not the
empty set, furthermore it is generally assumed that $N$ has more than two
processors. If $N$ has only one processor then load balancing is trivial as there is
only one processor to execute tasks. In the one processor case, this implementation
is almost identical to EEVDF, which is proposed in [1, 2]. In the two-processor
case, the load balancer is always considering the entire state of the system, which would allow for several simplifications in the implementation and in the proofs of the error bounds. Typically $n$ is used to denote an individual processor.

\[ w(t) - \text{The weight of a task } t. \quad (3.5) \]

\[ p(n) - \text{The performance of the given processor } n. \quad (3.6) \]

\[ w(t) \] is the weight of a given task $t$. The weight of a task is the task’s mostly commonly referenced property. This is same definition of task weight that is used by other proportional share schedulers [1–5]. Weight and performance are the two parameters the load balancer is ultimately trying to balance.

\[ p(n) - \text{The performance factor of the given processor } n. \quad (3.7) \]

\[ P - \text{The total performance of the system.} \]

\[ P = \sum_{n \in \mathbb{N}} p(n) \quad (3.8) \]

The function $p(n)$ is the performance of the given processor $n$, as a natural number. For this implementation, the Linux kernel’s BogoMIPS measurement is used for $p(n)$, but any empirical measurement could be substituted. Selecting an optimal method for empirically measuring the performance of a processor is beyond the scope of this paper. The load balancer proposed by this paper only requires that the function return a positive integer that grows linearly with the processor’s performance. $p(n) = 0$ for some processor $n$ would imply that the processor is incapable of performing any task, and thus it is assumed that $p(n) > 0$ holds for all processors.

$P$ is the total performance of all processors in the system. The value of $P$ can change while the system is running, since many system have CPU frequency
scaling which reduces the performance of the processor to save power. Most of the 
load balancer’s calculations only require information from the processors in the 
pair selected by the load balancer. \( P \) is one of two values the load balancer needs 
that require knowledge of the state of the entire system. The function \( w(t) \) is the 
weight of the given task \( t \). Weights are always integers and are strictly positive. A 
task with a weight of zero would essentially be requesting no processing time. 
The only situation in which a task should be requesting no time is if it is blocked 
or otherwise not running in which case it is not assigned to any processor.

\[
W(T) - \text{the weight of a set of tasks.}
\]

\[
W(T) = \sum_{t \in T} w(t) \quad (3.9)
\]

\[
(3.10)
\]

\( W(T) \) is the total of the weight of all tasks in the set \( T \), thus \( W(T) \) is 0 when \( T \) 
is empty. For any pair of disjoint sets of tasks \( M, N \), the function \( W(T) \) is 
distributive over the operation of set union. In other word for any two sets of task 
\( M \) and \( N \), if \( M \) and \( N \) have no common tasks then \( W(N \cup M) = W(T) + W(M) \). 
\( W(T) \) is also distributive over set-difference if and only if \( M \subseteq N \) such that 
\( W(N \setminus M) = W(T) - W(M) \).

**Lemma 3.3.1.** *The function \( W(T) \) is distributive over set union for all disjoint sets of 
tasks \( K, L \).*
Proof. of Lemma 3.3.1

\[ W(K) + W(L) \]
\[ \sum_{t \in K} w(t) + \sum_{t \in L} w(t) \]
\[ \sum_{t \in K \cup L} w(t) \]
\[ W(K \cup L) \]

□

**Lemma 3.3.2.** The function \( W(T) \) is distributive over set difference for all sets of tasks \( K, L \) s.t. \( K \subseteq L \).

Proof. of Lemma 3.3.2

\[ W(K) - W(L) \]
\[ \sum_{t \in K} w(t) - \sum_{t \in L} w(t) \]
\[ \sum_{t \in K \setminus L} w(t) - \sum_{t \in L \setminus K} w(t) \]
\[ L \subseteq K \implies L \setminus K = \emptyset \implies \sum_{t \in L \setminus K} w(t) = 0 \]
\[ \sum_{t \in K \setminus L} w(t) \]
\[ W(K \setminus L) \]

□

Adding tasks to a processor can be thought of as taking the set union of the tasks currently assigned to the processor and the set of tasks being moved to the processor. These two set are always disjoint, since a task cannot be added to processor if it is already on that processor. Similarly, removing task can be
thought of as the set-difference of the tasks assigned to a processor and a subset of those tasks that are being removed. The Lemmas 3.3.1 and 3.3.2 are used to represent these concepts when analyzing the effects of task movement on the error metrics later in this section.

\[ Q(n) - \text{the set of task on processor } n. \]  
\[ W(Q(n)) - \text{the total weight of all tasks on processor } n. \]  
\[ S - \text{the total weight of all tasks in the system.} \]

\[ S = \sum_{n \in N} W(Q(n)) \]  
\[ (3.12) \]

\[ (3.13) \]

\( Q(n) \) is the set of all proportional share tasks currently assigned to processor \( n \) from the set \( N \). \( W(Q(k)) \) then is the total weight of all tasks assigned to processor \( k \) often referred to as the weight of processor \( k \). \( Q(n) \) will be empty if the processor has no proportional share tasks and thus the processor will have a total weight of 0. If the performance of \( n \) is much smaller than the performance of the other processors in \( N \), \( Q(n) = \emptyset \) may be part of the optimal solution as in example 3.3. A general-purpose scheduler with the goal of maximizing the usage of system resources would never want to leave a processor idle when there is work that could be done. A proportional share scheduler also tries to use all of the system resources, but must also consider fairness. Since all tasks in the system must be on a processor, all task in the system can be denoted as \( \cup_{n \in N} Q(n) \).

**Example 3.3.** Assume a system has three tasks \( t_1, t_2, t_3 \) with the weights 5, 5, 5, and three processors \( k, l, j \) with performance values 100, 5000, 6000. Any task assigned to processor \( k \) will receive far fewer performance units per second than the tasks assigned to processors
l, and j. Let’s examine the error of assigning $t_1$ to k and $t_2$ to l and $t_3$ to j. Then $Q(k) = t_1$, $Q(l) = t_2$, and $Q(j) = t_3$ yielding the following total weights $W(Q(k)) = 5$, $W(Q(l)) = 5$, and $W(Q(j)) = 5$. The load on each processor will then be $r_k(Q(k)) = \frac{5(11100)}{100(15)} - 1 = 36.00$, $r_l(Q(l)) = \frac{5(11100)}{5000(15)} - 1 = -0.26$, and $r_j(Q(j)) = \frac{5(11100)}{6000(15)} - 1 = -0.38$. The load values show that processor k is 3600% overloaded, l is 26% under loaded, and j is 38% under loaded.

Giving a maximum error, R, of 36.00 and sum of errors, E, of 36.64. Moving $t_1$ to j will yield $Q(k) = \{\}$, $Q(l) = \{t_2\}$, and $Q(j) = \{t_1, t_3\}$, with loads of $r_k(Q(k)) = -1$, $r_l(Q(l)) = -0.26$, and $r_j(Q(j)) = .23$. Now the system error metrics will be $R = 1$ and $E = 1.49$.

\[
R = \max_{n \in N} |r_n(Q(n))| \quad (3.14)
\]
\[
E = \sum_{n \in N} |r_n(Q(n))| \quad (3.15)
\]
\[
r_n(T) = \frac{W(T)p}{p(n)S} - 1 \quad (3.16)
\]
\[
\alpha_n = \frac{p(n)}{p} \quad (3.17)
\]

$r_n(T)$ is the signed relative error of the given processor $n$ also referred to as the load function. A negative relative error indicates that a processor is under loaded, while a positive relative error indication that a processor is over loaded. It should also be noted that an alternative function is sometimes used for $r_n(T)$ and is equivalent to the above expression. The alternate $r_n(T)$ uses the function $\alpha_n$ which is the percentage of the total system performance represented by the given processor $n$. 
The alternative form for the function $r_n(T)$ demonstrates what the function actually represents. First, start with the original PSS proposal in [2]. In the PSS proposal each task ideally receive a share of the time determined by its weight specifically $w(t)\frac{\tau}{W(T)}$ where $\tau$ is the total time and $T$ is the set of all tasks. An SMP system though can be considered to have 1 second per processor of processing time for each second of actual time thus each task should receive the following amount of time $w(t)\frac{|N|\tau}{S}$ where $|N|$ is the number of processors and $S$ is the total weight of all tasks on all processors. In an asymmetric system, time is not a fair way to divide the resources, so instead of time the total performance of the system is used. In an asymmetric system each task should receive $w(t)\frac{P}{S}$ where $P$ is the total performance of the system.

Ideally, each processor will have a total weight of $\alpha_nS$. Since each task is assigned to a processor, a given task $t$ will receive an amount of performance equal to $w(t)\frac{P(n)}{W(Q(n))}$. If each processor has a total weight of $\alpha_nS$ then each task will receive performance of $w(t)\frac{P(n)}{\alpha_nS} = w(t)\left(\frac{P(n)}{S}\right)\frac{P}{\alpha_nS} = w(t)\frac{P}{S}$ which is the ideal amount.

Now consider the $r_n(Q(n)) = \frac{W(Q(n)) - \alpha_nS}{\alpha_nS}$. The numerator is simply the difference between the weight on processor $n$ and the ideal weight for processor $n$, while the denominator is the ideal weight. Thus the ratio of the two is simply the
percentage difference from the ideal weight for processor \( n \). When processor \( n \) has more than its ideal weight the ratio will be positive, and when it has less than its ideal weight the ratio will be negative.

The two system metrics \( R \) and \( E \) are simply aggregates of the function \( r_n(Q(n)) \). \( R \) is the maximum \( |r_n(Q(n))| \) for any processor in the system and thus represent the worst case tasks in the system, but gives no information about the rest of the system. \( E \) is the summation of \( |r_n(Q(n))| \) for each processor in the system and gives an idea of how much error is in the system as a whole, but give no information about how that error is distributed.

The last thing to note about these functions is their ranges of possible values. They are as follows:

\[
\begin{align*}
  w(t) &: (0, \infty) \\
  W(T) &: [0, \infty) \\
  S &: [0, \infty) \\
  p(n) &: (0, \infty) \\
  P &: (0, \infty) \\
  \alpha_n &: (0, 1] \\
  r_n(Q(n)) &: [-1, \infty) \\
  E &: [0, \infty) \\
  R &: [0, \infty)
\end{align*}
\]

3.3.2 Performance Model

Most Proportional Share Schedulers define fairness in terms of the amount of processing time received by a task [1, 2, 4, 5, 24]. In an asymmetric multiprocessor
machine, it is not fair to allocate processing resources simply in terms of time, since each processor performs differently.

\[ h_i(\tau) = \frac{w(t)}{S} \quad (3.19) \]

A Proportional Share Scheduler divides the systems processing time amongst the tasks such that each task receives \( h_i(\tau) \) weight at any time \( \tau \) in the ideal system [2]. In a single processor system this can be achieved by simply allocating \( h_i(\tau) \) of the only processor’s time to each task \( t \), as demonstrated in example 3.4. PS schedulers always use the entire system, so if the system has multiple processors, the tasks will have to be allocated amongst those processor such that each task is still being granted its fair share of the system. Example demonstrates such an allocation for a symmetric system.

**Example 3.4.** Assume single processor system has 4 tasks \( t_1, t_2, t_3 \) and \( t_4 \) and that these tasks have a weight of 2, 3, 7, and 8 respectively. Then the system has a total weight of 20. Ideally on any interval in which all 4 tasks are requesting processing time, task \( t_1 \) will receive \( \frac{2}{20} \) or 10% of the processors time. Tasks \( t_2, t_3, \) and \( t_4 \) should receive 15%, 35%, and 40% respectively.

If a fifth task, \( t_5 \), with a weight of 5 is later added then the allocation will change to 8%, 12%, 28%, 32%, and 20% for tasks \( t_1, t_2, t_3, t_4, \) and \( t_5 \) respectively.

**Example 3.5.** Assume a two processor system has the same 4 tasks as example 3.4, \( w(t_1) = 2, w(t_2) = 3, w(t_3) = 7, \) and \( w(t_4) = 8 \). If both processors have the same performance then a fair allocation would be to place tasks \( t_1 \) and \( t_4 \) onto one of the processors and tasks \( t_2 \) and \( t_3 \) onto the other processor. In this allocation is receiving \( \frac{2}{2+8} = 20\% \) of one processor and task \( t_4 \) is likewise receiving \( \frac{8}{2+8} = 80\% \) of the same processor. Tasks \( t_2 \) and \( t_3 \) are receiving 30% and 70% of their processor respectively. Since
each processor is 50% of the system, tasks $t_1$, $t_2$, $t_3$, and $t_4$ are receiving 10%, 15%, 35%, and 40% of the system respectively.

In order to provide a fair allocation processing resources, the load balancer uses performance units instead of time. Each processor has a performance function $p(n)$ which is the number of performance units the processor in $n$ produces in a seconds. Thus if a task $t$ runs on processor $n$ for $\tau$ seconds it will receive $\tau p(n)$ performance units. In the ideal fluid flow system [2], each task would receive a share of the total system performance equal to the ratio of its weight to the total system weight. Thus derived from equation (1) in [2] each task $t$ should receive the following instantaneous share of performance $f_t(\tau)$ at time $\tau$, where $S$ is total weight of all tasks in the system at time $\tau$ and $P$ is the total performance of the system at time $\tau$.

$$f_t(\tau) = \frac{w(t)}{S} P$$

(3.20)

Therefore from equation (2) in [2], on any interval $[\tau_0, \tau_1]$ during which the total weight and total performance of the system remain constant should receive total performance units of $\rho_t(\tau_0, \tau_1)$.

$$\rho_t(\tau_0, \tau_1) = \int_{\tau_0}^{\tau_1} f_t(x) dx$$

(3.21)

In a real system, tasks may only run on one processor at any given time. To handle this, the system assigns tasks to a particular processor, where each task competes only with the other tasks on that processor. Thus the instantaneous share for a task $t$ assigned to a processor $n$ at time $\tau$ is $g_{n,t}(\tau)$. 
\[
g_{n,t}(\tau) = \frac{w(t)}{W(Q(n))} p(n) \tag{3.22}
\]

If \( W(Q(N)) = \alpha_n S \), then \( g_{n,t}(\tau) = f_{\tau} \).

Thus verifying that the ideal weight assignment for any processor \( n \) is \( \alpha_n S \).

**Example 3.6.** Assume a two processor system with processors \( k \) and \( l \) has the same 4 tasks as example 3.5, \( w(t_1) = 2, w(t_2) = 3, w(t_3) = 7, \) and \( w(t_4) = 8 \). In this example the processors will have different performance values, let \( p(k) = 500 \) and \( p(l) = 1000 \), so processor \( l \) is twice as fast as processor \( k \). Since the system has a total weight of \( S = 2 + 3 + 7 + 8 = 20 \), processor \( k \) has an ideal weight of \( \alpha_k S = \frac{500}{1000+500} \times 20 = 6.66 \) and processor \( l \) has an ideal weight of \( \alpha_l S = 13.33 \). The ideal allocation can not be achieved since, fractional weights are not possible. An optimal allocation would be for task \( t_3 \) to be assigned to processor \( k \), and for tasks \( t_1, t_2, \) and \( t_4 \) to be assigned to processor \( l \). This yields \( Q(W(k)) = 7 \) and \( Q(W(l)) = 13 \). In this assignment task \( t_1 \) will receive a performance of \( g_{t_1,1}(\tau) = \frac{2}{13} \times 1000 = 154 \) when it should receive \( f_{t_1}(\tau) = \frac{2}{20} \times 1500 = 150 \). In terms of percentage task \( t_1 \) is receiving 15.4\% of processor \( l \) which is 10.3\% of the system, when it should be receiving 10.0\% under the ideal assignment. Tasks \( t_2, t_3, \) and \( t_4 \) are receiving 15.4\%, 33.3\%, and 41.0\% respectively, and would receive 15.0\%, 35.0\%, and 40.0\% in an ideal system.
However, if \( W(Q(N)) \neq \alpha_n S \), then tasks assigned to processor \( n \) will lag. Lag is the amount of performance a task is not receiving due to poor task assignment that it would be receiving in the ideal environment. In this paper lag will be measured as the ratio of performance lost to the amount of performance allocated in the ideal environment. Example demonstrates an allocation that is not ideal.

\[
\begin{align*}
\text{lag}_{n,t}(\tau) &= \frac{f_{n,t}(\tau) - g_{n,t}(\tau)}{f_{n,t}(\tau)} \\
\text{lag}_{n,t}(\tau) &= 1 - \frac{g_{n,t}(\tau)}{G_{n,t}(\tau)} \\
\text{lag}_{n,t}(\tau) &= 1 - \frac{p(n)w(t)}{W(Q(n))} \left( \frac{S}{w(t)P} \right) \\
\text{lag}_{n,t}(\tau) &= 1 - \frac{p(n)S}{W(Q(n))P} \\
\text{lag}_{n,t}(\tau) &= 1 - \frac{1}{\frac{p(n)S}{W(Q(n))P}} - 1 + 1 \\
\text{lag}_{n,t}(\tau) &= 1 - \frac{1}{r_n(Q(n)) + 1}
\end{align*}
\]

This measure of lag is independent of the individual tasks, and could be written as \( \text{lag}_n(\tau) \) instead of \( \text{lag}_{n,t}(\tau) \). Additionally, the lag can be computed from the same error function that the metrics use to determine the overall system error. A processor measure of lag is used, because the lag caused to individual task by time quanta is small compared to the lag caused by inaccuracies in the weight assignment.

**Theorem 3.3.3.** \( \text{lag}_{n'} < \text{lag}_n \) if and only if \( r_n'(Q(n')) < r_n(Q(n)) \), where \( n' \) is the state of processor \( n \) after some movement of tasks.

**Proof.** Proof of Theorem 3.3.3
Case 1: \( \text{lag}_{n'} < \text{lag}_n \) if \( r_{n'}(Q(n')) < r_n(Q(n)) \)

Assume that \( r_{n'}(Q(n')) < r_n(Q(n)) \) is true.

\[
\begin{align*}
    r_{n'}(Q(n')) &< r_n(Q(n)) \\
    r_{n'}(Q(n')) + 1 &< r_n(Q(n)) + 1 \\
    \frac{1}{r_{n'}(Q(n')) + 1} &< \frac{1}{r_n(Q(n)) + 1} \\
    1 - \frac{1}{r_{n'}(Q(n')) + 1} &< 1 - \frac{1}{r_n(Q(n)) + 1} \\
    \text{lag}_{n'} &< \text{lag}_n
\end{align*}
\]

Case 2: \( \text{lag}_{n'} < \text{lag}_n \) only if \( r_{n'}(Q(n')) < r_n(Q(n)) \)

Assume that \( r_{n'}(Q(n')) < r_n(Q(n)) \) is not true.

\[
\begin{align*}
    r_{n'}(Q(n')) &\geq r_n(Q(n)) \\
    r_{n'}(Q(n')) + 1 &\geq r_n(Q(n)) + 1 \\
    \frac{1}{r_{n'}(Q(n')) + 1} &\geq \frac{1}{r_n(Q(n)) + 1} \\
    1 - \frac{1}{r_{n'}(Q(n')) + 1} &\geq 1 - \frac{1}{r_n(Q(n)) + 1} \\
    \text{lag}_{n'} &\geq \text{lag}_n
\end{align*}
\]

\[\square\]

**Theorem 3.3.4.** \( \text{lag}_{n'} > \text{lag}_n \) if and only if \( r_{n'}(Q(n')) > r_n(Q(n)) \), where \( n' \) is the state of processor \( n \) after some movement of tasks.

*Proof.* Proof of Theorem 3.3.5
Case 1: \( \text{lag}_{n'} > \text{lag}_n \) if \( r_{n'}(Q(n')) > r_n(Q(n)) \)

Assume that \( r_{n'}(Q(n')) > r_n(Q(n)) \) is true.

\[
\begin{align*}
    r_{n'}(Q(n')) &> r_n(Q(n)) \\
    r_{n'}(Q(n')) + 1 &> r_n(Q(n)) + 1 \\
    \frac{1}{r_{n'}(Q(n')) + 1} &< \frac{1}{r_n(Q(n)) + 1} \\
    1 - \frac{1}{r_{n'}(Q(n')) + 1} &< 1 - \frac{1}{r_n(Q(n)) + 1} \\
    \text{lag}_{n'} &> \text{lag}_n
\end{align*}
\]

Case 2: \( \text{lag}_{n'} > \text{lag}_n \) only if \( r_{n'}(Q(n')) > r_n(Q(n)) \)

Assume that \( r_{n'}(Q(n')) > r_n(Q(n)) \) is not true.

\[
\begin{align*}
    r_{n'}(Q(n')) &\leq r_n(Q(n)) \\
    r_{n'}(Q(n')) + 1 &\leq r_n(Q(n)) + 1 \\
    \frac{1}{r_{n'}(Q(n')) + 1} &\geq \frac{1}{r_n(Q(n)) + 1} \\
    1 - \frac{1}{r_{n'}(Q(n')) + 1} &\geq 1 - \frac{1}{r_n(Q(n)) + 1} \\
    \text{lag}_{n'} &\leq \text{lag}_n
\end{align*}
\]

\[\square\]

**Theorem 3.3.5.** \( \text{lag}_{n'} = \text{lag}_n \) if and only if \( r_{n'}(Q(n')) = r_n(Q(n)) \), where \( n' \) is the state of processor \( n \) after some movement of tasks.

**Proof.** Proof of Theorem 3.3.5
Case 1: \( \text{lag}_{n'} = \text{lag}_n \) if \( r_{n'}(Q(n')) = r_n(Q(n)) \)

Assume that \( r_{n'}(Q(n')) = r_n(Q(n)) \) is true.

\[
\begin{align*}
\frac{1}{r_{n'}(Q(n')) + 1} &= \frac{1}{r_n(Q(n)) + 1} \\
1 - \frac{1}{r_{n'}(Q(n')) + 1} &= 1 - \frac{1}{r_n(Q(n)) + 1}
\end{align*}
\]

\( \text{lag}_{n'} = \text{lag}_n \)

Case 2: \( \text{lag}_{n'} = \text{lag}_n \) only if \( r_{n'}(Q(n')) > r_n(Q(n)) \)

Assume that \( r_{n'}(Q(n')) = r_n(Q(n)) \) is not true.

\[
\begin{align*}
\frac{1}{r_{n'}(Q(n')) + 1} &
\neq \frac{1}{r_n(Q(n)) + 1} \\
1 - \frac{1}{r_{n'}(Q(n')) + 1} &
\neq 1 - \frac{1}{r_n(Q(n)) + 1}
\end{align*}
\]

\( \text{lag}_{n'} \neq \text{lag}_n \)

3.3.3 Infeasible Weight Assignments

Other PS schedulers define an infeasible weight assignment as any scenario where there exists a task in the system that is requesting more processing time than 1 processor can provide, that is \( \frac{w(t)}{5} > \frac{1}{|N|} \) for some task \( t \).
Example 3.7. Assume a system has four processors with equal performance. Also assume that the system has four tasks, \(t_1, t_2, t_3,\) and \(t_4\) with weight 15, 12, 12, and 11. In this example the task \(t_1\) is infeasible, since it is requesting 30% of the processing time, \(\frac{15}{15+12+12+11} = 0.30\), but no single processor can provide more than 25%.

Where as an asymmetric system considers a task infeasible if there is no processor in the system that can service the task, which does not already, have a task with greater or equal weight. Many multiprocessor PS systems must reassign weights to resolve infeasible weight assignments [4, 5, 24]; however this load balancer does not perform any weight reassignments. Infeasible weight assignments do not exist in the single processor case, thus each individual processor simply schedules its tasks as normal. The load balancer is only concerned with the amount of overload or under load on processor, so an infeasible weight assignment simply meaning that some processor will be overloaded. As the load balancer tries to reduce the error it will tend to push the infeasible tasks to the higher performance processors.

Example 3.8. Assume a system has four processors \(k, l, j,\) and \(i\) with the following performance values, 14, 14, 11, and 11 for a total of 50 performance. Now assume there are four task \(t_1, t_2, t_3,\) and \(t_4\) with weight 15, 12, 12, and 11 for a total weight of 50. Task \(t_1\) is infeasible and one of tasks \(t_2\) and \(t_3\) is infeasible, but the other is feasible. Task \(t_1,\) the heaviest task, is infeasible since the fastest processor, \(k,\) only has a performance of 140. \(\frac{15}{50} > \frac{14}{50}\) or simply 0.30 > 0.28 in order to make a best effort attempt at fairness task \(t_1\) should get processor \(k\) to itself. Now consider tasks \(t_2\) and \(t_3\) which both have a weight of 12. Processor \(l\) is fast enough to run one of them fairly but not both, and neither processors \(j\) nor \(i\) are fast enough to run either \(t_2\) or \(t_3.\)
3.3.4 Criteria Implied by \( E \)

The following criteria are required for the load balancer to never increase error, \( E \). For all cases it is assumed that the load balancer is trying to increase the weight of \( k \) and decrease the weight of \( l \), for some selected pair of processors \( k \) and \( l \).

**Lemma 3.3.6.** Moving a set of tasks \( Y \) from processor \( l \) to processor \( k \) while also moving a set of tasks \( X \) from processor \( k \) to processor \( l \) will improve the error, \( E \), if and only if the move improves the sum of the errors on processors \( k \) and \( l \) is improved. \( E \) will improve iff

\[
|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| < |r_k(Q(k))| + |r_l(Q(l))|
\]

**Proof.** Lemma 3.3.6

Let \( E \) be the error prior to the move, then \( E = \sum_{n \in N} |r_n(Q(n))| \).

Let \( E' \) be the error after the move, then

\[
E' = \sum_{n \in N} |r_n(Q(n))| - |r_k(Q(k))| - |r_l(Q(l))| + |r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)|,
\]

where \( Y \subseteq Q(l) \) is the set of task being moved from processor \( l \) to processor \( k \),

\( X \subseteq Q(k) \) is the set of tasks being moved from processor \( k \) to processor \( l \),

\( r_n(Q(n)) \) is the relative error when the set of tasks \( Q(n) \) is run on processor \( n \),

\( Q(n) \) is the set of tasks currently running on processor \( n \), and

\( r_n(Q(n)) \) is thus the current relative error of processor \( n \).

Proof by cases first assuming

\[
|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| < |r_k(Q(k))| + |r_l(Q(l))|,
\]

then assuming it is false.
**Case 1:** $|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| < |r_k(Q(k))| + |r_l(Q(l))|$

$$E' = \left[ \sum_{n \in \mathbb{N}} |r_n(Q(n))| \right] - |r_k(Q(k))| - |r_l(Q(l))| + |r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)|$$

$$= \left[ \sum_{n \in \mathbb{N}} |r_n(Q(n))| \right] + (|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)|) - (|r_k(Q(k))| + |r_l(Q(l))|)$$

$$E' < E$$

**Case 2:** $|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| \geq |r_k(Q(k))| + |r_l(Q(l))|$

$$E' = \left[ \sum_{n \in \mathbb{N}} |r_n(Q(n))| \right] - |r_k(Q(k))| - |r_l(Q(l))| + |r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)|$$

$$= \left[ \sum_{n \in \mathbb{N}} |r_n(Q(n))| \right] + (|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)|) - (|r_k(Q(k))| + |r_l(Q(l))|)$$

$$\geq \left[ \sum_{n \in \mathbb{N}} |r_n(Q(n))| \right]$$

$$E' \geq E$$

\[\Box\]

**Theorem 3.3.7.** Moving a set of tasks $Y$ from processor $l$ to processor $k$, while also moving a set of tasks $X$ from processor $l$ to processor $k$ will reduce the sum of the absolute relative error for the pair $(k, l)$, if and only if at least one of the following criteria is true:
1. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus Y \cup X) > 0 \) and \( p(l) \leq p(k) \)

2. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus Y \cup X) > 0 \) and \( p(l) > p(k) \) and \( W(Y) - W(X) < 2 \left( \frac{\alpha \mu p(l) S - p(l) W(Q(k))}{p(l) - p(k)} \right) \)

3. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) \leq 0 \) and \( r_l(Q(l) \setminus Y \cup X) > 0 \)

4. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus Y \cup X) \leq 0 \) and \( W(Y) - W(X) < 2 \frac{p(k) W(Q(k)) - p(l) W(Q(l))}{p(k) + p(l)} \)

5. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) \leq 0 \) and \( r_l(Q(l) \setminus Y \cup X) \leq 0 \) and \( p(k) \leq p(l) \)

6. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) \leq 0 \) and \( r_l(Q(l) \setminus Y \cup X) \leq 0 \) and \( p(k) > p(l) \) and \( W(Y) - W(X) < 2 \frac{p(k) W(Q(k)) - p(l) W(Q(l))}{p(k) + p(l)} \)

7. \( r_k(Q(k)) > 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus Y \cup X) > 0 \) and \( p(k) > p(l) \)

8. \( r_k(Q(k)) > 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus Y \cup X) \leq 0 \) and \( W(Y) - W(X) < \frac{2 p(k) W(Q(k)) - 2 p(l) W(Q(l))}{p(k) + p(l)} \)

9. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) \leq 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus Y \cup X) \leq 0 \) and \( W(Y) - W(X) < \frac{2 \alpha \mu p(l) S - 2 p(l) W(Q(k))}{p(k) + p(l)} \)

10. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) \leq 0 \) and \( r_k(Q(k) \setminus X \cup Y) \leq 0 \) and \( r_l(Q(l) \setminus Y \cup X) \leq 0 \) and \( p(l) > p(k) \)

The proof of theorem 3.3.7 is provided in appendix A.
3.3.5 Simplifying Values that Occur in the Criteria for $E$

The criteria presented in Theorem 3.3.7, which are necessary for a move to reduce the system error metric $E$, look rather complicated and arbitrary. There are 10 separate criteria, and if any one of them is true then the move will reduce the error. There are so many criteria, because the lemma is covering all possible cases in an asymmetric system, but in any given scenario, the criteria reduce considerably. Before demonstrating how the criteria reduce in various scenarios, it is important to understand what the parts of the criteria mean. From Theorem 3.3.7 criteria 2, 4, 6, 8, and 9 all contain limits on $W(Y) - W(X)$, the meaning of which is not entirely obvious.

Each of the criteria 2, 4, 6, 8, 9 can have a 2 factored out of it. The 2 comes from the fact that load balancer is permitted to over adjust so long as the absolute value of the error is reduced. Thus, what was a negative error can become a positive of error of nearly the same magnitude. If $\omega$ is the amount of weight required to minimize then error then moving $2\omega$ will often just shift which processor has the error.

The criteria 2 include the value $2(\alpha_k S - W(Q(k)))\left(\frac{p(\cdot)}{p(\cdot) - p(k)}\right)$, which can be rewritten as $2\left(\alpha_k S - W(Q(k))\right)\left(\frac{p(l)}{p(l) - p(k)}\right)$. The 2 has already been explained as affect of allowing the load balancer to over adjust. The next part ($\alpha_k S - W(Q(k))$) has two terms $\alpha_k S$ and $W(Q(k))$. $\alpha_k S$ is the ideal amount of weight for processor $k$, and $W(Q(k))$ is the amount of weight actually assigned to processor $k$. $(\alpha_k S - W(Q(k)))$ is then the amount of weight that should be added to processor $k$. Since $k$ is under loaded, $(\alpha_k S - W(Q(k)))$ must be positive. The remaining term $\left(\frac{p(l)}{p(l) - p(k)}\right)$ is a factor to adjust for the difference in performance between $k$ and $l$ and is best understood when looking at criteria 1 and 2 together. Criteria 1 tries to move the entire overload to processor $k$, this is because processor $k$ is faster and the faster processor suffer less
relative overload for the same amount of weight. Criteria 2 only happens when processor l is faster, but since k starts under loaded some weight is still moved to k. The cost of moving extra weight to processor k is dependent on how much faster l is then k. \((\frac{p(l)}{p(l)-p(k)}) = 1\) when \(p(k) = 0\), processor k is much slower then processor l, but \(\lim_{p(k) \to p(l)} \frac{p(l)}{p(l)-p(k)} = \infty\), when k is almost as fast as l the value is large.

Figure 3.3: The above graphs shows the weights where criteria 2a1 or 2a2 will be true. Processor k is under loaded by 1, and l is overload by a large amount.

The criteria 9 has the value \(2\alpha S - W(Q(l))(\frac{p(l)}{p(k)+p(l)})\) which is similar to the value in criteria 2. \(2\alpha S - W(Q(l))(\frac{p(l)}{p(k)+p(l)})\) can be factor as \(2(\alpha S - W(Q(k)))(\frac{p(l)}{p(k)+p(l)})\). The only difference between the value in criteria 2 and this one is that the term \((\frac{p(l)}{p(l)-p(k)})\) is replaced with \((\frac{p(k)}{p(k)+p(l)})\). The term \((\frac{p(l)}{p(k)+p(l)})\) is the percentage of the performance of pair \{k, l\} that is contributed by processor l. When processors k and l have the same performance \((\frac{p(l)}{p(k)+p(l)}) = \frac{1}{2}\).
The values in the criteria 6 and 8 are identical to those in criteria 2 and 9 except that the roles of the processors are reversed. The value in 6, $2\left(\frac{p(k)W(Q(l)) - p(l)\alpha_lS}{p(k) - p(l)}\right)$, factors into $2(W(Q(l)) - \alpha_lS)(\frac{p(k)}{p(k) - p(l)})$. Instead of the under load on processor $k$ this value has the overload on processor $l$. Recall that the proof of Theorem 3.3.7 had numerous absolute values; the terms in the criteria have simply been ordered such that they are always positive when the other parts of the criteria are true. The value in criteria 8, $\frac{2p(l)W(Q(l)) - 2p(k)\alpha_lS}{p(k) + p(l)}$, factors into $2(W(Q(l)) - \alpha_lS)(\frac{p(k)}{p(k) + p(l)})$. This is similar to $2(\alpha_kS - W(Q(k)))(\frac{p(l)}{p(k) + p(l)})$ from criteria 9. The under load of $k$ is replaced with the overload of processor $l$, and the percentage performance of $l$ is replaced by the percentage performance of $k$.

Finally the value $2\left(\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}\right)$ from criteria 4. This value is different from the others in that it contains neither of the variables $P$ nor $S$. More significantly, on a two processor system only criteria 3 and 4 can be true. All of the other criteria either start or end with both processors being under loaded or overloaded. By the nature of PS systems there is always exactly the right amount of weight, thus a processor can be overloaded if and only if there is an under loaded processor in the system. The value $2\left(\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}\right)$ does not appear to contain the error of either processor but actually it contains the error from both.

The numerator is simply the sum of the $p(k)|W(Q(l)) - \alpha_lS|$ and $p(l)|W(Q(k)) - \alpha_kS|$.

$$p(k)|W(Q(l)) - \alpha_lS| + p(l)|W(Q(k)) - \alpha_kS|$$

$$p(k)[W(Q(l)) - \alpha_lS] + p(l)[\alpha_kS - W(Q(k))]$$

$$p(k)W(Q(l)) - \frac{1}{p}p(k)p(l) + \frac{1}{p}p(k)p(l) - p(l)W(Q(k))$$

$$p(k)W(Q(l)) - p(l)W(Q(k))$$
Thus $2 \left( \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \right)$ from criteria 4 is equivalent to

$$2 \left( \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \right) = \left( \frac{2}{p(k) + p(l)} \right) (p(k)|W(Q(l)) - \alpha_l S| + p(l)|W(Q(k)) - \alpha_k S|)$$

(3.23)

It is also worth noting that the numerator $p(k)W(Q(l)) - p(l)W(Q(k))$ is the cross product of inequality used to determine if processor $l$ is more loaded than processor $k$, $\frac{W(Q(l))}{p(k)} < \frac{W(Q(k))}{p(l)}$.

### 3.3.6 Simplifying the Criteria for $E$

Theorem 3.3.7 lists 10 criteria, each of which is a series of statements joined by logical ands. If any of the criteria are true, then committing the considered move will reduce the error metric $E$. In this section the criteria from 3.3.7 will be used to determine from any starting state the amount of weight the may be moved to improve the error metric $E$. In Theorem 3.3.7 the amount of weight being moved is referred as $W(Y) - W(X)$, in this section the amount of weight is referred to as $\delta$. Additionally $\sigma_k$ is the amount of weight the processor $k$ is either overloaded or underloaded, and $\sigma_l$ is the amount of weight that processor $l$ is overloaded or underloaded.

$$\delta = W(Y) - W(X)$$

$$\sigma_k = W(k) - \alpha_k S$$

$$\sigma_l = W(l) - \alpha_l S$$

Each of the criteria in Theorem 3.3.7 starts with a statement about the load on processor $k$. $r_k(Q(k)) \leq 0$ requires $k$ to be either balanced or under loaded, while
$r_k(Q(k)) > 0$ requires processor $k$ to be overloaded. The second statement in each criteria make a similar claim about the starting state of processor $l$. The third and fourth statements then assert that the processors are either under loaded or balanced or overloaded in the end state. This statement however can be transformed into an equivalent statement about the start state. If a processor was under loaded before the move and is over loaded after the move, then weight adjustment of the move must have been greater than the under loaded weight of the processor. The following are use to transform the criteria and remove statements about the end state:

\[
\begin{align*}
r_k(Q(k)) &\leq 0 \land \delta > -\sigma_k \equiv r_k(Q(k)) \leq 0 \land W(Y) - W(X) > \alpha_kS - W(Q(k)) \\
&\equiv r_k(Q(k)) \leq 0 \land W(Y) - W(X) - \alpha_kS > 0 \\
&\equiv r_k(Q(k)) \leq 0 \land \frac{W(Q(k)) + W(Y) - W(X) - \alpha_kS}{\alpha_kS} > 0 \\
r_k(Q(k)) &\leq 0 \land \delta > -\sigma_k \equiv r_k(Q(k)) \leq 0 \land r_k(Q(k) \cup Y \setminus X) > 0 \quad (3.24)
\end{align*}
\]

\[
\begin{align*}
r_k(Q(k)) &\leq 0 \land \delta \leq -\sigma_k \equiv r_k(Q(k)) \leq 0 \land W(Y) - W(X) \leq \alpha_kS - W(Q(k)) \\
&\equiv r_k(Q(k)) \leq 0 \land W(Y) - W(X) - \alpha_kS \leq 0 \\
&\equiv r_k(Q(k)) \leq 0 \land \frac{W(Q(k)) + W(Y) - W(X) - \alpha_kS}{\alpha_kS} \leq 0 \\
r_k(Q(k)) &\leq 0 \land \delta \leq -\sigma_k \equiv r_k(Q(k)) \leq 0 \land r_k(Q(k) \cup Y \setminus X) \leq 0 \quad (3.25)
\end{align*}
\]
\[ r_k(Q(k)) > 0 \land \delta > 0 \implies r_k(Q(k)) > 0 \land W(Q(k)) - \alpha_kS > 0 \land W(Y) - W(X) > 0 \]
\[ \implies r_k(Q(k)) > 0 \land W(Q(k)) + W(Y) - W(X) - \alpha_kS > 0 \]
\[ \land W(Y) - W(X) > 0 \]
\[ \implies r_k(Q(k)) > 0 \land \frac{W(Q(k)) + W(Y) - W(X) - \alpha_kS}{\alpha_kS} > 0 \]
\[ \land W(Y) - W(X) > 0 \]
\[ r_k(Q(k)) > 0 \land \delta > 0 \implies r_k(Q(k)) > 0 \land r_k(Q(k) \cup Y \setminus X) > 0 \]
\[ \land W(Y) - W(X) > 0 \]

Clearly \( r_k(Q(k)) > 0 \land r_k(Q(k) \cup Y \setminus X) > 0 \land W(Y) - W(X) \implies r_k(Q(k)) > 0 \land \delta > 0. \]

\[ \therefore r_k(Q(k)) > 0 \land \delta > 0 \iff r_k(Q(k)) > 0 \land r_k(Q(k) \cup Y \setminus X) > 0 \]
\[ \land W(Y) - W(X) > 0 \quad (3.26) \]

Weight is being removed from processor \( l \), so the equivalencies for \( r_l \) are slightly different.

\[ r_l(Q(l)) > 0 \land \delta < \alpha_l \iff r_l(Q(l)) > 0 \land W(Y) - W(X) < W(Q(l)) - \alpha_lS \]
\[ \iff r_l(Q(l)) > 0 \land 0 < W(Q(l)) - W(Y) + W(X) - \alpha_lS \]
\[ \iff r_l(Q(l)) > 0 \land \frac{W(Q(l)) - W(Y) + W(X) - \alpha_lS}{\alpha_lS} > 0 \]
\[ \iff r_l(Q(l)) > 0 \land \delta < \alpha_l \iff r_l(Q(l)) > 0 \land r_l(Q(l) \setminus Y \cup X) > 0 \quad (3.27) \]

\[ r_l(Q(l)) > 0 \land \delta \geq \alpha_l \iff r_l(Q(l)) > 0 \land W(Y) - W(X) \geq w(Q(l)) - \alpha_lS \]
\[ \iff r_l(Q(l)) > 0 \land 0 \geq W(Q(l)) - W(Y) + W(X) - \alpha_lS \]
\[ \iff r_l(Q(l)) > 0 \land \frac{W(Q(l)) - W(Y) + W(X) - \alpha_lS}{\alpha_lS} \leq 0 \]
\[ \iff r_l(Q(l)) > 0 \land \delta \geq \alpha_l \iff r_l(Q(l)) > 0 \land r_l(Q(l) \setminus Y \cup X) \leq 0 \quad (3.28) \]
Corollary 3.3.8. Theorem 3.3.7 can be stated in terms of \( \delta, \sigma_k, \) and \( \sigma_l, \) with no references to the load of the processors after the weight is moved.

1. \( r_k(Q(k)) \leq 0 \land \delta > 0 \implies r_l(Q(l)) > 0 \land \delta > -\sigma_k \land \delta < \sigma_l \land p(l) \leq p(k) \)

2. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \land \delta > -\sigma_k \land \delta < \sigma_l \land p(l) > p(k) \land \delta < 2 \left( \frac{\sigma_k p(l) - p(l) W(Q(k))}{p(l) - p(k)} \right) \)

3. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \land \delta > -\sigma_k \land \delta < \sigma_l \)

4. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \land \delta > -\sigma_k \land \delta \geq \sigma_l \land \delta < 2 \left( \frac{\sigma_k p(l) - p(l) W(Q(k))}{p(k) + p(l)} \right) \)

5. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \land \delta \leq -\sigma_k \land \delta \geq \sigma_l \land p(k) \leq p(l) \)

6. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \land \delta \leq -\sigma_k \land \delta \geq \sigma_l \land p(k) > p(l) \land \delta < 2 \left( \frac{\sigma_k p(l) - p(l) W(Q(k))}{p(k) - p(l)} \right) \)

7. \( r_k(Q(k)) > 0 \land r_l(Q(l)) > 0 \land \delta < \sigma_l \land p(k) > p(l) \)

8. \( r_k(Q(k)) > 0 \land r_l(Q(l)) > 0 \land \delta \geq \sigma_l \land \delta < \frac{2 p(k) W(Q(k)) - 2 p(l) \sigma_l S}{p(k) + p(l)} \)

Using the above equivalences the criteria of Theorem 3.3.7 can be restated as:

\[ r_l(Q(l)) \leq 0 \land \delta > 0 \implies r_l(Q(l)) \leq 0 \land W(Y) - W(X) > 0 \land W(Q(l)) - \alpha_l S \leq 0 \]

\[ \implies r_l(Q(l)) \land W(Y) - W(X) > 0 \land W(Q(l)) - W(Y) + W(X) - \alpha_l S \leq 0 \]

\[ \implies r_l(Q(l)) \land W(Y) - W(X) > 0 \land \frac{W(Q(l)) - W(Y) + W(X) - \alpha_l S}{\alpha_l S} \leq 0 \]

\[ r_l(Q(l)) \leq 0 \land \delta > 0 \implies r_l(Q(l)) \leq 0 \land W(Y) - W(X) > 0 \land r_l(Q(l) \setminus Y \cup X) \leq 0 \]

Clearly,

\[ r_l(Q(l)) \leq 0 \land W(Y) - W(X) > 0 \land r_l(Q(l) \setminus Y \cup X) \leq 0 \implies r_l(Q(l)) \leq 0 \land \delta > 0. \]

\[ \therefore r_l(Q(l)) \leq 0 \land \delta > 0 \iff r_l(Q(l)) \leq 0 \land W(Y) - W(X) > 0 \land r_l(Q(l) \setminus Y \cup X) \leq 0 \]

(3.29)
9. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) \leq 0 \land \delta > -\sigma_k \land \delta < \frac{2\sigma_k p(l) - 2p(l) W(Q(k))}{p(k) + p(l)} \)

10. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) \leq 0 \land \delta \leq -\sigma_k \land p(l) > p(k) \)

Before further simplifying theorem 3.3.7 there are a few lemmas that will be useful.

**Lemma 3.3.9.** If \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \), then

\[ \sigma_l \geq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \implies 2 \left( \frac{p(k) W(l) - p(l) W(k)}{p(k) + p(l)} \right) \geq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \]

Let \( x = \frac{p(k)}{p(l)} \), then \( p(k) = xp(l) \).

\[ \sigma_l \geq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \]

\[ \sigma_l \geq -2\sigma_k \frac{p(l)}{p(l) - xp(l)} \]

\[ \sigma_l \geq -2\sigma_k \frac{1}{1 - x} \]

\[ (1 - x)\sigma_l \geq -2\sigma_k \]

\[ \sigma_l - x\sigma_l \geq -2\sigma_k \]

\[ x\sigma_l - x^2\sigma_l \geq -2x\sigma_k \]

\[ x\sigma_l - \sigma_k - x^2\sigma_l + x\sigma_k \geq -x\sigma_k - \sigma_k \]

\[ (x\sigma_l - \sigma_k)(1 - x) \geq -\sigma_k(x + 1) \]

\[ \frac{1}{p(l)}(xp(l)\sigma_l + p(l)(-\sigma_k))(1 - x) \geq -\sigma_k(x + 1)p(l) \]

\[ \frac{1}{p(l)} \left( \frac{1}{x + 1} (xp(l)\sigma_l + p(l)(-\sigma_k)) \right) \geq -\sigma_k \frac{1}{1 - x} \]

\[ \frac{2}{xp(l) + p(l)} (xp(l)\sigma_l + p(l)(-\sigma_k)) \geq -2\sigma_k \frac{p(l)}{p(l) - xp(l)} \]

\[ \frac{2}{p(k) + p(l)} (p(k)\sigma_l + p(l)(-\sigma_k)) \geq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \]

By equation 3.23

\[ 2 \left( \frac{p(k) W(l) - p(l) W(k)}{p(k) + p(l)} \right) \geq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \]
Lemma 3.3.10. If \( r_k(Q(k)) \leq 0 \) \& \( r_l(Q(l)) > 0 \), then
\[
\sigma_l \leq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \implies 2\frac{p(l)W(l) - p(l)W(k)}{p(l)} \leq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \]
Let \( x = \frac{p(k)}{p(l)} \), then \( p(k) = xp(l) \).
\[
\begin{align*}
\sigma_l & \leq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \\
\sigma_l & \leq -2\sigma_k \frac{p(l)}{p(l) - xp(l)} \\
\sigma_l & \leq -2\sigma_k \frac{1}{1 - x} \\
(1 - x)\sigma_l & \leq -2\sigma_k \\
\sigma_l - x\sigma_l & \leq -2\sigma_k \\
x\sigma_l - x^2\sigma_l & \leq -2x\sigma_k \\
x\sigma_l - \sigma_k - x^2\sigma_l + x\sigma_k & \leq -x\sigma_k - \sigma_k \\
(x\sigma_l - \sigma_k)(1 - x) & \leq -\sigma_k(x + 1) \\
\frac{1}{p(l)}(xp(l)\sigma_l + p(l)(-\sigma_k))(1 - x) & \leq -\sigma_k(x + 1) \\
\frac{1}{p(l)} \frac{1}{x + 1} (xp(l)\sigma_l + p(l)(-\sigma_k)) & \leq -\sigma_k \frac{1}{1 - x} \\
\frac{2}{xp(l) + p(l)}(xp(l)\sigma_l + p(l)(-\sigma_k)) & \leq -2\sigma_k \frac{p(l)}{p(l) - xp(l)} \\
\frac{2}{p(k) + p(l)}(p(k)\sigma_l + p(l)(-\sigma_k)) & \leq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \\
\end{align*}
\]
By equation 3.23
\[
2\left( \frac{p(k)W(l) - p(l)W(k)}{p(k) + p(l)} \right) \leq -2\sigma_k \frac{p(l)}{p(l) - p(k)}
\]
Lemma 3.3.11. If $r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0$, then

$$\sigma_l \geq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \implies \sigma_l \geq 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}$$

Let $x = \frac{p(k)}{p(l)}$, then $p(k) = xp(l)$.

$$\sigma_l \geq -2\sigma_k \frac{p(l)}{p(l) - xp(l)}$$

$$(1 - x)\sigma_l \geq -2\sigma_k p(l) \frac{1}{1 - x} (1 - x)$$

$$\sigma_l - x\sigma_l \geq -2\sigma_k$$

$$\sigma_l \geq x\sigma_l - 2\sigma_k$$

$$x\sigma_l + \sigma_l \geq 2x\sigma_l - 2\sigma_k$$

$$(x + 1)\sigma_l \geq 2(x\sigma_l - \sigma_k)$$

$$\sigma_l \geq \frac{2}{x + 1} (xW(Q(l)) - \alpha_l S + [\alpha_k S - W(Q(k))])$$

$$\sigma_l \geq \frac{2}{x + 1} \frac{p(l)}{p(l)} (xW(Q(l)) - xp(l)S + xp(l)S - W(Q(k)))$$

$$\sigma_l \geq \frac{2}{xp(l) + p(l)} (xp(l)W(Q(l)) - p(l)W(Q(k)))$$

$$\sigma_l \geq 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}$$
Lemma 3.3.12. If \( r_k(Q(k)) \leq 0 \) \& \( r_l(Q(l)) > 0 \), then

\[
\sigma_l \leq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \implies \sigma_l \leq 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \quad \text{Let } x = \frac{p(k)}{p(l)}, \text{ then } p(k) = xp(l).
\]

\[
\sigma_l \leq -2\sigma_k \frac{p(l)}{p(l) - xp(l)}
\]

\[
(1 - x)\sigma_l \leq -2\sigma_k p(l) \frac{1}{1 - x}(1 - x)
\]

\[
\sigma_l - x\sigma_l \leq -2\sigma_k
\]

\[
\sigma_l \leq x\sigma_l - 2\sigma_k
\]

\[
x\sigma_l + \sigma_l \leq 2x\sigma_l - 2\sigma_k
\]

\[
(x + 1)\sigma_l \leq 2(x\sigma_l - \sigma_k)
\]

\[
\sigma_l \leq \frac{2}{x + 1}(x[W(Q(l)) - \alpha_kS] + [\alpha_kS - W(Q(k))])
\]

\[
\sigma_l \leq \frac{2}{x + 1} \frac{p(l)}{p(l)}(xW(Q(l)) - xp(l)S + xp(l)S - W(Q(k)))
\]

\[
\sigma_l \leq \frac{2}{xp(l) + p(l)}(xp(l)W(Q(l)) - p(l)W(Q(k)))
\]

\[
\sigma_l \leq 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}
\]

3.3.6.1 Underloaded/Overloaded: Symmetric

To start consider a pair of processors in a system \( \{k, l\} \) such that \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \). Processor \( k \) is under loaded, processor \( l \) is overloaded, and the load balancer is trying to move weight to processor \( k \). This assumption considers the first 6 criteria of Corollary 3.3.8. The last four are false due to the assumption that \( k \) is underloaded and \( l \) is overloaded. Next lets make the assumption that the pair is symmetric so \( p(k) = p(l) \). Then criteria 2 and 6 must be false, leaving only 4
criteria to consider. Note the when \( p(k) = p(l) \) that
\[
2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} = W(l) - W(k).
\]

Using the above assumptions the criteria can be rewritten as follows:

1. \( \delta > -\sigma_k \land \delta < \sigma_l \) which is \( -\sigma_k < \delta < \sigma_l \).

2. false.

3. \( \delta \leq -\sigma_k \land \delta < \sigma_l \).

4. \( \delta > -\sigma_k \land \delta \geq \sigma_l \land \delta < W(Q(l)) - W(Q(k)) \).

5. \( \delta - \sigma_k \land \delta \geq \sigma_l \) which is \( \sigma_l \leq \delta \leq -\sigma_k \).

6. false.

Finally, consider the two cases \( -\sigma_k < \sigma_l \) and \( -\sigma_k \geq \sigma_l \).

**Case 1: \( -\sigma_k < \sigma_l \)**

With this additional assumption the criteria simplify to the following:

1. \( -\sigma_k < \delta < \sigma_l \)

2. false.

3. \( \delta \leq -\sigma_k \) since \( \delta \geq -\sigma_k \land -\sigma_k < \sigma_l \implies \delta < \sigma_l \)

4. \( \delta \geq \sigma_l \land \delta < W(Q(l)) - W(Q(k)) \) since \( \delta \geq \sigma_l \land -\sigma_k < \sigma_l \implies \delta > -\sigma_k \)
5. false.
6. false.

Now the above criteria can simply be combined using logical ‘or’.

\[ (-\sigma_k < \delta < \sigma_l) \lor (\delta \leq -\sigma_k) \lor (\delta \geq \sigma_l \land \delta < W(Q(l)) - W(Q(k))) \]
\[ (\delta < \sigma_l) \lor (\delta \geq \sigma_l \land \delta < W(Q(l)) - W(Q(k))) \]
\[ \delta < W(Q(l)) - W(Q(k)) \]

Therefore if \( l \) has more error, by weight, than \( k \) and the pair is symmetric then the criteria for \( k \) underloaded and \( l \) overloaded become \( \delta < W(Q(l)) - W(Q(k)) \).

**Case 2: \(-\sigma_k \geq \sigma_l\)**

With this additional assumption the criteria simplify to the following:

1. false.
2. false.
3. \( \delta < \sigma_l \) since \( \delta < \sigma_l \land -\sigma_k \geq \sigma_l \implies \delta \leq -\sigma_k \)
4. \( \delta > -\sigma_k \land \delta < W(Q(l)) - W(Q(k)) \) since \( \delta \geq \sigma_l \land -\sigma_k < \sigma_l \implies \delta > -\sigma_k \)
5. \( \sigma_l \leq \delta \leq -\sigma_k \).
6. false.

Again the above criteria can simply be combined using logical ‘or’.

\[ \delta < \sigma_l \lor \delta > -\sigma_k \land \delta < W(Q(l)) - W(Q(k)) \lor \sigma_l \leq \delta \leq -\sigma_k \]
\[ \delta > -\sigma_k \land \delta < W(Q(l)) - W(Q(k)) \lor \delta \leq -\sigma_k \]
\[ \delta < W(Q(l)) - W(Q(k)) \]
Thus regardless of which processor has more error if a the pair of processors is symmetric and $k$ is under loaded while $l$ is overloaded, the criteria for the error metric $E$ can be reduced to $\delta < W(Q(l)) - W(Q(k))$. It is also worth noting that $W(Q(l)) - W(Q(k)) = \sigma_l - \sigma_k$, in this case.

3.3.6.2 Underloaded/Overloaded: Overloaded is Faster

Now keep the assumptions that $r_k(Q(k)) \leq 0$ and $r_l(Q(l)) > 0$, but consider the case where $l$ is faster than $k$, $p(k) < p(l)$. criteria 1 and 6 are false, since they require that processor $k$ be at least as fast as processor $l$. Under this set of assumptions, the criteria can be rewritten as follows:

1. false.

2. $\delta > -\sigma_k \wedge \delta < \sigma_l \wedge \delta < -2\sigma_k \frac{p(l)}{p(l)-p(k)}$.

3. $\delta \leq -\sigma_k \wedge \delta < \sigma_l$.

4. $\delta > -\sigma_k \wedge \delta \geq \sigma_l \wedge \delta < 2\frac{p(k)W(Q(l))-p(l)W(Q(k))}{p(k)+p(l)}$.

5. $\delta \leq -\sigma_k \wedge \delta \geq \sigma_l$.

6. false.

Next $\delta$ is considered in three different cases based on $\sigma_l$. All three cases however, will have the same result.

**Case 1:** $\sigma_l \leq -\sigma_k$

Criteria 2 states that $-\sigma_k < \delta < \sigma_l$ must hold, but it is false in this case. The criteria become:

1. false.

2. false.
3. \( \delta < \sigma \) since \( \sigma \leq -\sigma \) \( \wedge \) \( \delta < \sigma \) \( \implies \) \( \delta \leq -\sigma \).

4. \( \delta > -\sigma \) \( \wedge \) \( \delta < 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \) since \( \sigma \leq -\sigma \) \( \wedge \) \( \delta > -\sigma \) \( \implies \) \( \delta \geq \sigma \).

5. \( \sigma \leq \delta \leq -\sigma \).

6. false.

\( \sigma \leq -\sigma \) also implies that \( \sigma \leq -2\sigma \frac{p(l)}{p(l) - p(k)} \), since \( -\sigma > 0 \) and \( 2\frac{p(l)}{p(l) - p(k)} > 1 \).

The criteria can now be combined using logical 'or' as follows:

\[
(\delta < \sigma) \lor (\sigma \leq \delta \leq -\sigma) \lor \left( \delta > -\sigma \land \delta < 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \right)
\]

\[
(\delta < \sigma) \lor (\sigma \leq \delta \leq -\sigma) \lor \left( \delta > -\sigma \land \delta < 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \right)
\]

\[
(\delta \leq -\sigma) \lor \left( \delta < 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \right)
\]

\[
\delta < 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}
\]

The last step removes the statement \( \delta \leq -\sigma \), this is valid, since

\(-\sigma < 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \) is true. To see that that \( 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} > -\sigma \) is true, consider equation \( 3.23 \).

\[
2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} = \left( \frac{2}{p(k) + p(l)} \right) (p(k)\sigma_i - p(l)\sigma_k)
\]

Since \( p(k) < p(l) \),

\[
\frac{2}{p(k) + p(l)} > \frac{2}{p(l) + p(l)} > \frac{2}{p(l) + p(l)} \left( p(k)\sigma_i - p(l)\sigma_k \right)
\]

\[
> \frac{2}{p(l) \sigma_i - \sigma_k}
\]

\[
> 2 \left( \frac{2p(l)}{p(l) \sigma_i - \sigma_k} \right)
\]

\[
> -2\sigma_k
\]

\[
> -\sigma_k
\]
Therefore, \(2^{p(k)W(Q(l)) - p(l)W(Q(k))}/p(k) + p(l)} > -\sigma_k \) is true.

By lemma 3.3.10, \(2^{p(k)W(Q(l)) - p(l)W(Q(k))}/p(k) + p(l)} \geq -2\sigma_k^{p(l)}{p(k) - p(k)},\) thus the bounds for \(\delta\) can also be expressed as \(\delta < \min(-2\sigma_k^{p(l)}{p(k) - p(k)}, 2^{p(k)W(Q(l)) - p(l)W(Q(k))}/p(k) + p(l)}\).

**Case 2:** \(-2\sigma_k^{p(l)}{p(k) - p(k)} \geq \sigma_l > -\sigma_k\)

When processor \(l\) has more error, by weight, than \(k\) the criteria become:

1. false.

2. \(-\sigma_k < \delta < \sigma_l \land \delta < -2\sigma_k^{p(l)}{p(k) - p(k)}\).

3. \(\delta \leq -\sigma_k\), since \(\sigma_l > -\sigma_k \land \delta \leq -\sigma_k \implies \delta < \sigma_l\).

4. \(\delta \geq \sigma_l \land \delta < -2\sigma_k^{p(k)W(Q(l)) - p(l)W(Q(k))}/p(k) + p(l)}\), since \(\delta \geq \sigma_l \land \delta > -\sigma_k \implies \delta > -\sigma_k\).

5. false, since \(\delta \leq -\sigma_k \land \delta \geq \sigma_l \implies \sigma_l < -\sigma_k\).

6. false.

Additionally, \(-2\sigma_k^{p(l)}{p(k) - p(k)} \geq \sigma_l\) allows the following change to criteria 2.

1. false.

2. \(-\sigma_k < \delta < \sigma_l\).

3. \(\delta \leq -\sigma_k\).

4. \(\delta \geq \sigma_l \land \delta < 2^{p(k)W(Q(l)) - p(l)W(Q(k))}/p(k) + p(l)}\).

5. false.

6. false.
The logical 'or' of the above criteria can be reduced to:

\[
(\delta \leq -\sigma_k) \vee (-\sigma_k < \delta < \sigma_l) \vee \left( \delta \geq \sigma_l \land \delta < 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \right)
\]

\[
\delta < \sigma_l \lor \delta < 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}
\]

By lemma 3.3.12 \( \sigma_l \leq 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \).

By lemma 3.3.10, \( 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \leq -2\sigma_k \frac{p(l)}{p(l) - p(k)} \), therefore the bounds on \( \delta \) can be expressed as \( \delta < \min(-2\sigma_k \frac{p(l)}{p(l) - p(k)} , 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}) \).

**Case 3:** \( \sigma_l > -2\sigma_k \frac{p(l)}{p(l) - p(k)} \).

This case assumes the previous cases are false and that the error by weight on \( l \) is greater than \( -2\sigma_k \frac{p(l)}{p(l) - p(k)} \). Now all but two of the criteria are false.

1. false.

2. \( -\sigma_k < \delta < -2\sigma_k \frac{p(l)}{p(l) - p(k)} \), since \( \delta < -2\sigma_k \frac{p(l)}{p(l) - p(k)} \Rightarrow \delta < \sigma_l \).

3. \( \delta \leq -\sigma_k \).

4. false, since by lemma 3.3.11 \( \sigma_l \geq 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \).

5. false.

6. false.

The logical or of the remaining criteria is:

\[
(\delta \leq -\sigma_k) \vee (-\sigma_k < \delta < -2\sigma_k \frac{p(l)}{p(l) - p(k)})
\]

\[
\delta < -2\sigma_k \frac{p(l)}{p(l) - p(k)}
\]
By Lemma 3.3.9, \(2^{p(l)W(Q(l)) - p(k)W(Q(k))}/p(k) + p(l)} > -2\sigma_k p(l)/p(k) - p(k),\) therefore the bounds on \(\delta\) can also be expressed as \(\delta < \min(-2\sigma_k, 2^{p(l)W(Q(l)) - p(k)W(Q(k))})\) in this case.

All three cases result in \(\delta < \min(-2\sigma_k, 2^{p(l)W(Q(l)) - p(k)W(Q(k))})\). Thus if \(r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \land p(k) < p(l),\) then any \(\delta < \max(\sigma_k, 2^{p(l)W(Q(l)) - p(k)W(Q(k))})\) will reduce the metric \(E\).

### 3.3.6.3 Underloaded/Overloaded: Underloaded is Faster

Still holding the assumptions that \(r_k(Q(k)) \leq 0\) and \(r_l(Q(l)) > 0\), consider the case where \(k\) is faster than \(l\), \(p(k) > p(l)\). Now the load balancer is trying to move weight from the slower processor to the faster processor. In the scenario criteria 2 and 5 are false, since they both imply that processor \(l\) is the faster processor.

1. \(-\sigma_k < \delta < \sigma_l\)
2. false, since \(p(l) > p(k)\) is false.
3. \(\delta \leq -\sigma_k \land \delta < \sigma_l\)
4. \(\delta > -\sigma_k \land \delta \geq \sigma_l \land \delta < \frac{2p(k)}{p(k) + p(l)}(W(Q(l)) - p(l)W(Q(k)))\)
5. false, since \(p(k) \leq p(l)\) is false.
6. \(\sigma_l \leq \delta < -\sigma_k \land \delta < \left(\frac{2p(k)}{p(k) + p(l)}\right)\sigma_l\)

The remainder of this demonstration is divided into the following cases:

- \(-\sigma_k < \sigma_l\)
- \(\sigma_l \leq -\sigma_k < \left(\frac{2p(k)}{p(k) - p(l)}\right)\sigma_l\)
- \(\sigma_l < \left(\frac{2p(k)}{p(k) - p(l)}\right)\sigma_l \leq -\sigma_k \land \left(\frac{2}{p(k) + p(l)}\right)(p(k)W(Q(l)) - p(l)W(Q(l))) \leq -\sigma_k\)
- \(\sigma_l < \left(\frac{2p(k)}{p(k) - p(l)}\right)\sigma_l \leq -\sigma_k < \left(\frac{2}{p(k) + p(l)}\right)(p(k)W(Q(l)) - p(l)W(Q(l)))\)
Each case implies all of the previous case are false, thus only one case can be true for any given situation.

**Case 1:** $-\sigma_k < \sigma_l$

1. $-\sigma_k < \delta < \sigma_l$
2. false.
3. $\delta \leq -\sigma_k$, since $\delta \leq -\sigma_k \land -\sigma_k < \sigma_l \implies \delta < \sigma_l$.
4. $\delta \geq \sigma_l \land \delta < 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}$, since $\delta > \sigma_l \implies \delta > -\sigma_k$.
5. false.
6. false, since $\sigma_l \leq \delta \leq -\sigma_k$.

The logical 'or' of the criteria is:

\[
(\delta \leq -\sigma_k) \lor (-\sigma_k < \delta < \sigma_l) \lor \left(\delta \geq \sigma_l \land \delta < 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}\right)
\]

**Case 2:** $\sigma_l \leq -\sigma_k < 2\sigma_l\left(\frac{p(k)}{p(k) - p(l)}\right)$

1. false, since $-\sigma_k < \delta < \sigma_l$ is false.
2. false.
3. $\delta < \sigma_l$, since $\delta < \sigma_l \implies \delta \leq -\sigma_k$.
4. $\delta > -\sigma_k \land \delta < 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}$, since $\delta > -\sigma_k \implies \delta \geq \sigma_l$.
5. false.
6. $\sigma_l \leq \delta \leq -\sigma_k$ since $\delta \leq -\sigma_k \Rightarrow \delta < \left( \frac{2p(k)}{p(k) - p(l)} \right) \sigma_l$.

The logical or of the criteria is:

$$(\delta < \sigma_l) \lor (\sigma_l \leq \delta \leq -\sigma_k) \lor \left( \delta > -\sigma_k \land \delta < 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k)))}{p(k) + p(l)} \right) \frac{2p(k)W(Q(l)) - p(l)W(Q(k)))}{p(k) + p(l)}$$

**Case 3:** $2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right) \leq -\sigma_k$

The first step in this case is to show that $2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right) \leq -\sigma_k$ implies

$$2 \left( \frac{(p(k)W(Q(l)) - p(l)W(Q(k)))}{p(k) + p(l)} \right) \leq -\sigma_k.$$
Let \( x = \frac{p(0)}{p(l)} \), then \( p(l) = xp(k) \).

\[
-\sigma_k \geq 2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right)
\]

\[
-\sigma_k \geq 2\sigma_l \left( \frac{p(k)}{p(k) - xp(k)} \right)
\]

\[
-\sigma_k \geq 2\sigma_l \left( \frac{1}{1-x} \right)
\]

\[
(1-x)(-\sigma_k) \geq 2\sigma_l p(k)
\]

Since \( p(k) \geq 1 \)

\[
-\sigma_k + x\sigma_k \geq 2\sigma_l
\]

\[
-\sigma_k \geq 2\sigma_l - x\sigma_k
\]

\[
-x\sigma_k - \sigma_k \geq 2\sigma_l - 2x\sigma_k
\]

\[
(x + 1)(-\sigma_k) \geq 2(\sigma_l - x\sigma_k)
\]

Since \( r_l(Q(l)) > 0 \land r_k(Q(k)) \leq 0 \)

\[
-\sigma_k \geq \left( \frac{2}{x+1} \right) (x[\alpha_kS - W(Q(k))] + [W(Q(l)) - \alpha_lS])
\]

\[
-\sigma_k \geq \left( \frac{2}{x+1} \right) \left( \frac{p(k)}{p(k)} \right) (xp(k) S \frac{1}{p} - xW(Q(k)) + W(Q(l)) - xp(k) S \frac{1}{p})
\]

\[
-\sigma_k \geq \left( \frac{2}{xp(k) + p(k)} \right) (-xp(k)W(Q(k)) + p(k)W(Q(k)))
\]

\[
-\sigma_k \geq 2 \left( \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \right)
\]

1. false, since \(-\sigma_k \geq 2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right) > \sigma_l \).

2. false.

3. \( \delta < \sigma_l \)

4. false, since \( 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \leq -\sigma_k \)

5. false.
6. \( \sigma_l \leq \delta < 2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right) \)

This logical ‘or’ is:

\[
(\delta < \sigma_l) \lor \left( \sigma_l \leq \delta < 2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right) \right)
\]

\[
\delta < 2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right)
\]

Similarly to the previous part, these three cases can be combined into a single criterion. Cases 1 and 2 have the same result, but case 3 has a different criteria.

The assumption of case 3, however imply a relation between the limits on \( \delta \) in cases 1 and 2 and case 3. The following demonstrates that when the assumption of case 3 is false that

\[
2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right) > 2\frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}.
\]

Let \( x = \frac{p(l)}{p(k)} \), then \( p(l) = xp(k) \).

\[
2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right) > -\sigma_k
\]

\[
2\sigma_l \left( \frac{p(k)}{p(k) - xp(k)} \right) > -\sigma_k
\]

\[
2\sigma_l \left( \frac{1}{1 - x} \right) > -\sigma_k
\]

\[
2\sigma_l > (1 - x)(-\sigma_k)
\]

\[
2\sigma_l > -\sigma_k + x\sigma_k
\]

\[
\sigma_l - x\sigma_k + \sigma_l > -\sigma_k
\]
Since $r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0$

\[
\sigma_l + x \left[ p(k) \frac{S}{P} - W(Q(k)) \right] + \left[ W(Q(l)) - xp(k) \frac{S}{P} \right] > -\sigma_k
\]

\[
\sigma_l + \left[ xp(k) \frac{S}{P} - xW(Q(k)) + W(Q(l)) - xp(k) \frac{S}{P} \right] > -\sigma_k
\]

\[
\sigma_l + W(Q(l)) - xW(Q(k)) > -\sigma_k
\]

\[
\sigma_l + W(Q(l)) - xW(Q(k)) > -\sigma_k
\]

Since $r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0$

\[
\left[ W(Q(l)) - xp(k) \frac{S}{P} \right] + W(Q(l)) - xW(Q(k)) > \left[ p(k) \frac{S}{P} - W(Q(k)) \right]
\]

\[
xW(Q(l)) - x^2p(k) \frac{S}{P} + xW(Q(l)) - x^2W(Q(k)) > xp(k) \frac{S}{P} - xW(Q(k)) \]

\[
W(Q(l)) + xW(Q(l)) - x^2p(k) \frac{S}{P} + xW(Q(l)) - x^2W(Q(k)) > W(Q(l)) + xp(k) \frac{S}{P} - xW(Q(k))
\]

\[
W(Q(l)) - xp(k) \frac{S}{P} + xW(Q(l)) - x^2p(k) \frac{S}{P} > W(Q(l)) - xW(Q(k)) - xW(Q(l)) + x^2W(Q(k))
\]

\[
(1 + x)(W(Q(l)) - xp(k) \frac{S}{P}) \geq (1 - x)(W(Q(l)) - xW(Q(k)))
\]

\[
(W(Q(l)) - xp(k) \frac{S}{P}) \left( \frac{1}{1 - x} \right) > \frac{W(Q(l)) - xW(Q(k))}{1 + x}
\]

\[
(W(Q(l)) - xp(k) \frac{S}{P}) \left( \frac{p(k)}{p(k) - xp(k)} \right) > \frac{p(k)W(Q(l)) - xp(k)W(Q(k))}{p(k) + xp(k)}
\]

\[
(W(Q(l)) - p(l) \frac{S}{P}) \left( \frac{p(k)}{p(k) - p(l)} \right) > \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}
\]

Since $r_l(Q(l)) > 0$

\[
2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right) > 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}
\]
The next proof demonstrates that when the assumption of case 3 is true, that

\[ 2 \sigma_l \left( \frac{p(k)}{p(l)} \right) \leq 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) - p(l)}. \]

Let \( x = \frac{p(l)}{p(k)} \), then \( p(l) = xp(k) \).

\[
2 \sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right) \leq - \sigma_k \\
2 \sigma_l \left( \frac{p(k)}{p(k) - xp(k)} \right) \leq - \sigma_k \\
2 \sigma_l \left( \frac{1}{1 - x} \right) \leq - \sigma_k \\
2 \sigma_l \leq (1 - x)(- \sigma_k) \\
2 \sigma_l \leq - \sigma_k + x \sigma_k \\
\sigma_l - x \sigma_k + \sigma_l \leq - \sigma_k
\]

Since \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \)

\[
\sigma_l + x \left[ p(k) \frac{S}{P} - W(Q(k)) \right] + \left[ W(Q(l)) - xp(k) \frac{S}{P} \right] \leq - \sigma_k \\
\sigma_l + \left[ xp(k) \frac{S}{P} - xW(Q(k)) + W(Q(l)) - xp(k) \frac{S}{P} \right] \leq - \sigma_k \\
\sigma_l + W(Q(l)) - xW(Q(k)) \leq - \sigma_k \\
\sigma_l + W(Q(l)) - xW(Q(k)) \leq - \sigma_k
\]

Since \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \)

\[
\left[ W(Q(l)) - xp(k) \frac{S}{P} \right] + W(Q(l)) - xW(Q(k)) \leq \left[ p(k) \frac{S}{P} - W(Q(k)) \right] \\
xW(Q(l)) - x^2 p(k) \frac{S}{P} + xW(Q(l)) - x^2 W(Q(k)) \leq xp(k) \frac{S}{P} - xW(Q(k)) \\
W(Q(l)) + xW(Q(l)) - x^2 p(k) \frac{S}{P} + xW(Q(l)) - x^2 W(Q(k)) \leq W(Q(l)) + xp(k) \frac{S}{P} - xW(Q(k))
\]
\[ W(Q(l)) - xp(k) \frac{S}{P} + xW(Q(l)) - x^2p(k)\frac{S}{P} \leq W(Q(l)) - xW(Q(k)) - xW(Q(l)) + x^2W(Q(k)) \]

\[ (1 + x)(W(Q(l)) - xp(k)\frac{S}{P}) \leq (1 - x)(W(Q(l)) - xW(Q(k))) \]

\[ (W(Q(l)) - xp(k)\frac{S}{P}) \left( \frac{1}{1 - x} \right) \leq \frac{W(Q(l)) - xW(Q(k))}{1 + x} \]

\[ (W(Q(l)) - xp(k)\frac{S}{P}) \left( \frac{p(k)}{p(k) - xp(k)} \right) \leq \frac{p(k)W(Q(l)) - xp(k)W(Q(k))}{p(k) + xp(k)} \]

\[ (W(Q(l)) - p(l)\frac{S}{P}) \left( \frac{p(k)}{p(k) - p(l)} \right) \leq \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \]

Since \( r_l(Q(l)) > 0 \)

\[ 2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right) \leq 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \]

Since it is now known how the values \( 2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right) \) and \( 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \) are related in each case, the three cases can be combined into the following single criteria.

\[ \delta < \min \left( 2\sigma_l \left( \frac{p(k)}{p(k) - p(l)} \right), 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \right) \]

### 3.3.6.4 Both Overloaded

This situation is covered by criteria 7 and 8. Weight is moved to the faster processor, when both processors are overloaded. When processor \( k \) is the faster processor, \( p(k) > p(l) \), the criteria are reduced to:

7. \( \delta < \sigma_l \).

8. \( \delta \geq \sigma_l \land \delta < \left( \frac{2p(k)}{p(k) + p(l)} \right) \sigma_l \).

Since \( k \) is faster than \( l \), \( \left( \frac{2p(k)}{p(k) + p(l)} \right) > 1 \). Thus when combined the criteria becomes

\[ \delta < \left( \frac{2p(k)}{p(k) + p(l)} \right) \sigma_l \].
Now consider when \( l \) is faster than \( k \). In this case the criteria become:

7. false.

8. \( \delta \geq \sigma_l \land \delta < \left( \frac{2p(k)}{p(k) + p(l)} \right) \sigma_l \).

This case can only be true if delta is in the range \( \sigma_l \) and \( \left( \frac{2p(k)}{p(k) + p(l)} \right) \sigma_l \). Since \( l \) is faster \( \left( \frac{2p(k)}{p(k) + p(l)} \right) < 1 \), thus the range is empty and no value of \( \delta \) will make the criteria true.

In the both overloaded scenario weight should only be moved if \( k \) is the faster processor.

### 3.3.6.5 Both Underloaded

Criteria 9 and 10, apply when both processors are under loaded. If both processors are under loaded weight is moved to the slower processor. Consider the case where \( l \) is faster than \( k \), \( p(l) > p(k) \). Then the criteria become:

9. \( \delta > -\sigma_k \land \delta < -\sigma_k \left( \frac{2p(l)}{p(k) + p(l)} \right) \).

10. \( \delta \leq -\sigma_k \).

Since \( l \) is the faster processor, \( \left( \frac{2p(l)}{p(k) + p(l)} \right) > 1 \). When the are combined to yield \( \delta < -\sigma_k \left( \frac{2p(l)}{p(k) + p(l)} \right) \).

When processor \( k \) is the faster processor the criteria become:

9. \( \delta > -\sigma_k \land \delta < -\sigma_k \left( \frac{2p(l)}{p(k) + p(l)} \right) \).

10. false.

Similar to the both overloaded scenario, \( \left( \frac{2p(l)}{p(k) + p(l)} \right) < 1 \) when processor \( k \) is faster. The range in criteria 9 is thus empty, and the criteria always result in false. When both processors are under loaded weight is only moved to the slower processor.

Now criteria for \( \delta \) can be summarized into a table organized by the state of the system.
3.3.7 Optimal Choice of Weight for Metric $E$

The bounds presented in 3.3.6 give the maximum amount of weight the can be moved while still improving the error metric $E$. Additionally, any amount of weight not in the bounds will not improve the metric $E$. Thus any weight assignment where the bounds are empty is an optimal weight assignment for the given pair of processors.

**Lemma 3.3.13.** For any pair of processors, $i$ and $j$, if $p(i) \leq p(j)$ and $\sigma_i = 0$, then no improvement can be made to metric $E$ by attempting to further balance $i$ and $j$.

**Proof.** Proof of Lemma 3.3.13 For some pair of processors $i$ and $j$, assume $p(i) \leq p(j)$ and $\sigma_i = 0$. Let $Y$ and $X$ be set of tasks such that $X \subset Q(i)$ and $Y \subset Q(j)$. Let $\delta = W(Y) - W(X)$. Let $E'$ be the new value of the metric $E$ after the tasks in $X$ are moved to $j$ and the sets in $Y$ are moved to $i$, so that 

$$E' = E + |r_j(Q(j) \setminus Y \cup X)| + |r_i(Q(i) \setminus X \cup Y)| - |r_j(Q(j))| - |r_i(Q(i))|.$$ 

The following demonstrates the $E' > E$ for any pair of sets $Y$ and $X$. 

<table>
<thead>
<tr>
<th>rule</th>
<th>$r_i(Q(k)) \leq 0$</th>
<th>$r_l(Q(l)) \leq 0$</th>
<th>performance criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>true</td>
<td>false</td>
<td>$p(k) = p(l)$</td>
</tr>
<tr>
<td>2</td>
<td>true</td>
<td>false</td>
<td>$p(k) &lt; p(l)$</td>
</tr>
<tr>
<td>3</td>
<td>true</td>
<td>false</td>
<td>$p(k) &gt; p(l)$</td>
</tr>
<tr>
<td>4</td>
<td>false</td>
<td>false</td>
<td>$p(k) &gt; p(l)$</td>
</tr>
<tr>
<td>5</td>
<td>true</td>
<td>true</td>
<td>$p(k) &lt; p(l)$</td>
</tr>
</tbody>
</table>
\[ E' = E + |r_j(Q(j) \setminus Y \cup X)| + |r_i(Q(i) \setminus X \cup Y)| - |r_j(Q(j))| - |r_i(Q(i))| \]

Since \( \sigma_i = 0 \implies r_i(Q(i)) = 0. \)

\[ E' = E + \left\lvert \frac{W(Q(j)) - \delta P}{p(j)S} \right\rvert - 1 + \left\lvert \frac{W(Q(i)) + \delta P}{p(i)S} \right\rvert - 1 - \left\lvert \frac{W(Q(j))P}{p(j)S} \right\rvert - 1 - 0 \]

\[ E' = E + \left\lvert \frac{1}{p(j)S}[W(Q(j)) - \delta P - p(j)S] + \frac{1}{p(i)S}[W(Q(i)) + \delta P - p(i)S] - \frac{1}{p(j)S}W(Q(j))P - p(j)S \right\rvert \]

Since \( W(Q(i)) - \frac{p(i)S}{P} = \sigma_i = 0. \)

\[ E' = E + \frac{P}{p(j)S} |\sigma_j - \delta| + \frac{P}{p(i)S} |\delta| - \frac{P}{p(j)S} |\sigma_j| \]

Now \( E' \) is broken into 4 case to remove some of the absolute values.

**Case 1: \( \sigma_j \geq 0 \) and \( \delta \leq \sigma_j \)

\[ E' = E + \frac{P}{p(i)S} |\delta| + \frac{P}{p(j)S} \sigma_j - \frac{P}{p(j)S} \delta - \frac{P}{p(j)S} \sigma_j \]

\[ E' = E + \frac{P}{p(i)S} |\delta| - \frac{P}{p(j)S} \delta \]

Since \( p(i) \leq p(j) \implies \frac{P}{p(i)S} \geq \frac{P}{p(j)S} \implies \frac{P}{p(i)S} |\delta| - \frac{P}{p(j)S} \delta \geq 0. \)

\[ E' \geq E \]

**Case 2: \( \sigma_j \leq 0 \) and \( \delta \geq \sigma_j \)

\[ E' = E + \frac{P}{p(i)S} |\delta| - \frac{P}{p(j)S} \sigma_j + \frac{P}{p(j)S} \delta + \frac{P}{p(j)S} \sigma_j \]

\[ E' = E + \frac{P}{p(i)S} |\delta| + \frac{P}{p(j)S} \delta \]

Since \( p(i) \leq p(j) \implies \frac{P}{p(i)S} \geq \frac{P}{p(j)S} \implies \frac{P}{p(i)S} |\delta| + \frac{P}{p(j)S} \delta \geq 0. \)

\[ E' \geq E \]
Case 3: $\sigma_j \geq 0$ and $\delta > \sigma_j$

$$E' = E + \frac{p}{p(i)S}|\delta| - \frac{p}{p(j)S}\sigma_i + \frac{p}{p(j)S}\delta - \frac{p}{p(j)S}\sigma_j$$

Since $\delta > \sigma_j \geq 0$.

$$E' = E + \frac{p}{p(i)S}|\delta| - \frac{p}{p(j)S}\sigma_j$$

Since $p(i) \leq p(j) \land \delta > \sigma_j \geq 0 \implies \frac{p}{p(i)S}|\delta| - \frac{p}{p(j)S}\sigma_j \geq 0$.

$$E' \geq E$$

Case 4: $\sigma_j \leq 0$ and $\delta < \sigma_j$

$$E' = E + \frac{p}{p(i)S}|\delta| + \frac{p}{p(j)S}\sigma_j - \frac{p}{p(j)S}\delta + \frac{p}{p(j)S}\sigma_j$$

Since $\delta < \sigma_j \leq 0$.

$$E' = E + \frac{p}{p(i)S}|\delta| + \frac{p}{p(j)S}\sigma_j$$

Since $p(i) \leq p(j) \land \delta > \sigma_j \geq 0 \implies \frac{p}{p(i)S}|\delta| + \frac{p}{p(j)S}\sigma_j \geq 0$.

$$E' \geq E$$

\[ \square \]

Lemma 3.3.14. For any pair of processors, $i$ and $j$, if $p(i) = p(j)$ and $\frac{W(i) + W(j)}{p(i) + p(j)} > \frac{s}{p}$, then all $\delta$ in $[\!-\sigma_k, \sigma_l\!]$ produce the same value of $E$.

Proof. For some pair of processors $i$ and $j$, assume $p(i) = p(j)$ and $\frac{W(i) + W(j)}{p(i) + p(j)} > \frac{s}{p}$. Let $Y$ and $X$ be set of tasks such that $X \subset Q(i)$ and $Y \subset Q(j)$ and

$\delta = W(Y) - W(X) \in [\!-\sigma_k, \sigma_l\!]$. Let $E'$ be the new value of the metric $E$ after the tasks in $X$ are moved to $j$ and the sets in $Y$ are moved to $i$, so that
For any pair of processors, i and j, if \( p(i) = p(j) \) then all \( \delta \in [\sigma_i, -\sigma_i] \) produce the same value of \( E \).

**Lemma 3.3.15.** For any pair of processors, i and j, if \( p(i) = p(j) \) and \( \frac{w(i) + w(j)}{p(i) + p(j)} < \frac{s}{p} \), then \( E' > E \) for any pair of sets \( Y \) and \( X \).

**Proof.** For some pair of processors \( i \) and \( j \), assume \( p(i) = p(j) \) and \( \frac{w(i) + w(j)}{p(i) + p(j)} < \frac{s}{p} \). Let \( Y \) and \( X \) be set of tasks such that \( X \subset Q(i) \) and \( Y \subset Q(j) \) and \( \delta = W(Y) - W(X) \in [\sigma_j, -\sigma_i] \). Let \( E' \) be the new value of the metric \( E \) after the tasks in \( X \) are moved to \( j \) and the sets in \( Y \) are moved to \( i \), so that \( E' = E + |r_j(Q(j) \cup X)| + |r_i(Q(i) \cup Y)| - |r_j(Q(j))| - |r_i(Q(i))| \). The following demonstrates the \( E' > E \) for any pair of sets \( Y \) and \( X \).
\[ E' = E + |r_j(Q(j) \setminus Y \cup X)| + |r_i(Q(i) \setminus X \cup Y)| - |r_j(Q(j))| - |r_i(Q(i))| \]

\[ = E + \frac{|W(Q(j)) - W(Y) + W(X) - \alpha_j S|}{\alpha_j S} + \frac{|W(Q(i)) - W(X) + W(Y) - \alpha_i S|}{\alpha_i S} - |r_j(Q(j))| - |r_i(Q(i))| \]

\[ = E + \frac{\sigma_j - \delta}{\alpha_j S} + \frac{\sigma_i + \delta}{\alpha_i S} - |r_j(Q(j))| - |r_i(Q(i))| \]

Since \( \delta \leq -\sigma_i \), and since \( \delta \geq \sigma_j \).

Since \( p(i) = p(j) \), \( \alpha_i = \alpha_j \).

\[ E' = E + \frac{-\sigma_j - \sigma_i}{\alpha_i S} - |r_j(Q(j))| - |r_i(Q(i))| \]

Therefore the value of \( E' \) is independent of the value of \( \delta \) chosen from \( [\sigma_j, -\sigma_i] \). □

**Theorem 3.3.16.** The ideal value, \( \delta'_{E'} \), is present for each case in the following table.

<table>
<thead>
<tr>
<th>rule</th>
<th>performance</th>
<th>pair load</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( p(k) &lt; p(l) )</td>
<td>any</td>
<td>( -\sigma_k )</td>
</tr>
<tr>
<td>2</td>
<td>( p(k) &gt; p(l) )</td>
<td>any</td>
<td>( \sigma_l )</td>
</tr>
<tr>
<td>3</td>
<td>( p(k) = p(l) )</td>
<td>( \frac{W(Q(l)) + W(Q(k))}{p(l) + p(k)} &gt; \frac{s}{p} )</td>
<td>( [-\sigma_k, \sigma_l] )</td>
</tr>
<tr>
<td>4</td>
<td>( p(k) = p(l) )</td>
<td>( \frac{W(Q(l)) + W(Q(k))}{p(l) + p(k)} &lt; \frac{s}{p} )</td>
<td>( [\sigma_l, -\sigma_k] )</td>
</tr>
<tr>
<td>5</td>
<td>( p(k) = p(l) )</td>
<td>( \frac{W(Q(l)) + W(Q(k))}{p(l) + p(k)} = \frac{s}{p} )</td>
<td>( \sigma_l = -\sigma_k )</td>
</tr>
</tbody>
</table>

**Proof.** Proof of Theorem 3.3.16 This proof provides one case for each of the rule presented in Theorem 3.3.16.
**Case 1:** \( p(k) < p(l) \) and \( \delta_E^* = -\sigma_k \)

Let \( \sigma_k' \) be the error by weight on processor \( k \) after \( \delta_E^* \) weight has been moved to it.

\[
\begin{align*}
\sigma_k' &= W(Q(k)) + \delta_E^* - \alpha_k S \\
\sigma_k' &= W(Q(k)) - \sigma_k - \alpha_k S \\
\sigma_k' &= W(Q(k)) - W(Q(k)) - \alpha_k S - \alpha_k S \\
\sigma_k' &= 0
\end{align*}
\]

By lemma 3.3.13 moving \( -\sigma_k \) to processor \( k \) creates a state with the minimum value of metric \( E \) for this pair.

**Case 2:** \( p(k) > p(l) \) and \( \delta_E^* = \sigma_l \)

Let \( \sigma_l' \) be the error by weight on processor \( l \) after \( \delta_E^* \) weight has been moved away from processor \( l \).

\[
\begin{align*}
\sigma_l' &= W(Q(l)) - \delta_E^* - \alpha_l S \\
\sigma_l' &= W(Q(l)) - \sigma_l - \alpha_l S \\
\sigma_l' &= W(Q(l)) - W(Q(l)) - \alpha_l S - \alpha_l S \\
\sigma_l' &= 0
\end{align*}
\]

By lemma 3.3.13 moving \( \sigma_l \) away from processor \( l \) creates a state with the minimum value of metric \( E \) for this pair.

**Case 3:** \( p(k) = p(l) \) and \( \frac{W(Q(l)) + W(Q(k))}{p(l) + p(k)} > \frac{S}{2} \) and \( \delta_E^* \in [-\sigma_k, \sigma_l] \)

Let \( \sigma_l' \) be the error by weight on processor \( l \) after \( \sigma_l \) weight has been moved away
from processor $l$.

$$\sigma'_l = W(Q(l)) - \sigma_l - \alpha_l S$$

$$\sigma'_l = W(Q(l)) - W(Q(l)) - \alpha_l S - \alpha_l S$$

$$\sigma'_l = 0$$

By lemma 3.3.13 moving $\sigma_l$ away from processor $l$ creates a state with the minimum value of metric $E$ for this pair. By lemma 3.3.14 all values of $\delta^*_E$ produce the same value of $E$ as $\sigma_l$.

$$\therefore \text{ all } \delta^*_E \in [-\sigma_k, \sigma_l] \text{ are optimal.}$$

**Case 4:** $p(k) = p(l)$ and $\frac{W(Q(l)) + W(Q(k))}{p(l) + p(k)} < \frac{S}{2}$ and $\delta^*_E \in [\sigma_l, -\sigma_k]$  

Let $\sigma'_l$ be the error by weight on processor $l$ after $\sigma_l$ weight has been moved away from processor $l$.

$$\sigma'_l = W(Q(l)) - \sigma_l - \alpha_l S$$

$$\sigma'_l = W(Q(l)) - W(Q(l)) - \alpha_l S - \alpha_l S$$

$$\sigma'_l = 0$$

By lemma 3.3.13 moving $\sigma_l$ away from processor $l$ creates a state with the minimum value of metric $E$ for this pair. By lemma 3.3.15 all values of $\delta^*_E$ produce the same value of $E$ as $\sigma_l$.

$$\therefore \text{ all } \delta^*_E \in [\sigma_l, -\sigma_k] \text{ are optimal.}$$

**Case 5:** $p(k) = p(l)$ and $\frac{W(Q(l)) + W(Q(k))}{p(l) + p(k)} = \frac{S}{2}$ and $\delta^*_E = \sigma_l$  

Let $\sigma'_l$ be the error by weight on processor $l$ after $\delta^*_E$ weight has been moved away from processor $l$. 
\[
\sigma'_l = W(Q(l)) - \delta^*_E - \alpha_l S
\]
\[
\sigma'_l = W(Q(l)) - \alpha_l S
\]
\[
\sigma'_l = W(Q(l)) - W(Q(l)) - \alpha_l S - \alpha_l S
\]
\[
\sigma'_l = 0
\]

By lemma 3.3.13 moving \(\sigma_l\) away from processor \(l\) creates a state with the minimum value of metric \(E\) for this pair.

Also note that in this case \(-\sigma_k = \sigma_l\). To demonstrate the value of \(W(Q(k))\) is first determined.

\[
\frac{W(Q(l)) + W(Q(k))}{p(l) + p(k)} = \frac{S}{P}
\]

Since \(p(l) = p(k)\).

\[
W(Q(l)) + W(Q(k)) = \frac{2p(l)S}{P}
\]

\[W(Q(k)) = 2\alpha_l S - W(Q(l))\]

Now \(-\sigma_k\) is evaluated.

\[
-\sigma_k = \frac{-W(Q(k)) + \alpha_k S}{\alpha_k S} = \frac{-[2\alpha_l S - W(Q(l))] + \alpha_k S}{\alpha_k S} = \frac{W(Q(l)) - 2\alpha_l S + \alpha_k S}{\alpha_k S}
\]

Since \(p(l) = p(k)\), \(\alpha_l = \alpha_k\).

\[
-\sigma_k = \frac{W(Q(l)) - \alpha_l S}{\alpha_l S} = \frac{W(Q(l)) - 2\alpha_l S + \alpha_l S}{\alpha_l S}
\]

Therefore \(-\sigma_k\) is also optimal in this case. \(\Box\)
3.3.8 Criteria Implied by $R$

Lemma 3.3.17. For any pair of processors $k$ and $l$ and sets of tasks $Y \subseteq Q(l)$ and $X \subseteq Q(k)$ such that $W(Y) - W(X) \leq 0$ and assuming without loss of generality $r_k(Q(k)) \leq r_l(Q(l))$, then

$$\max(|r_k(Q(k)\setminus X \cup Y)|, |r_l(Q(l)\setminus Y \cup X)|) \geq \max(|r_k(Q(k))|, |r_l(Q(l))|).$$

Proof. of Lemma 3.3.17

Case 1: $r_k(Q(k)) = r_l(Q(l))$

Part 1: $r_k(Q(k)) = r_l(Q(l)) \geq 0$

$$r_l(Q(l)\setminus Y \cup X) = \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1$$

$$= \frac{-[W(Y) - W(X)]P}{p(l)S} + \frac{W(Q(l))P}{p(l)S} - 1$$

$$= \frac{-[W(Y) - W(X)]P}{p(l)S} + r_l(Q(l))$$

$$\geq r_l(Q(l)) \quad \text{by} \quad (W(Y) - W(X) \leq 0)$$

$$r_l(Q(l)\setminus Y \cup X) \geq r_l(Q(l)) \geq 0$$

$$|r_l(Q(l)\setminus Y \cup X)| \geq |r_l(Q(l))|$$

Therefore

$$\max(|r_k(Q(k)\setminus X \cup Y)|, |r_l(Q(l)\setminus Y \cup X)|) \geq \max(|r_k(Q(k))|, |r_l(Q(l))|)$$
Part 2: \( r_k(Q(k)) = r_i(Q(l)) \leq 0 \)

\[
\begin{align*}
r_k(Q(k) \setminus X \cup Y) &= \frac{[W(Q(k)) + W(Y) - W(X)]p}{p(k)S} - 1 \\
&= \frac{[W(Y) - W(X)]p}{p(k)S} + \frac{W(Q(k))p}{p(k)S} - 1 \\
&= \frac{[W(Y) - W(X)]p}{p(k)S} + r_k(Q(k)) \\
&\leq r_k(Q(k)) \quad \text{by } (W(Y) - W(X) \leq 0)
\end{align*}
\]

\[
\begin{align*}
r_k(Q(k) \setminus X \cup Y) &\leq r_k(Q(k)) \leq 0 \\
|r_k(Q(k) \setminus X \cup Y)| &\geq |r_k(Q(k))|
\end{align*}
\]

Therefore \( \max(|r_k(Q(k) \setminus X \cup Y)|, |r_i(Q(l) \setminus Y \cup X)|) \geq \max(|r_k(Q(k))|, |r_i(Q(l))|) \)

Essential this case states that if the pair of processors has equal error then no improvement can be made by moving tasks between them.

Case 2: \( |r_k(Q(k))| \geq |r_i(Q(l))| \)
Part 1: \( r_k(Q(k)) \leq 0 \)

\[
\begin{align*}
  r_k(Q(k) \setminus X \cup Y) &= \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \\
  &= \frac{[W(Y) - W(X)]P}{p(k)S} + \frac{W(k)P}{p(k)S} - 1 \\
  &= \frac{[W(Y) - W(X)]P}{p(k)S} + r_k(Q(k)) \\
  \leq & r_k(Q(k)) \quad \text{by } (W(Y) - W(X) \leq 0)
\end{align*}
\]

\( r_k(Q(k) \setminus X \cup Y) \leq r_k(Q(k)) \leq 0 \)

\[|r_k(Q(k) \setminus X \cup Y)| \geq |r_k(Q(k))| \geq 0\]

\[|r_k(Q(k) \setminus X \cup Y)| \geq \max(|r_k(Q(k))|, |r_l(Q(l))|)\]

therefore \( \max(|r_k(Q(k) \setminus X \cup Y)|, |r_l(Q(l) \setminus Y \cup X)|) \geq \max(|r_k(Q(k))|, |r_l(Q(l))|) \)

Part 2: \( r_k(Q(k)) \geq 0 \)

Since \( 0 \leq r_k(Q(k)) \leq r_l(Q(l)) \) and \( |r_k(Q(k))| \geq |r_l(Q(l))| \), it must hold that \( r_k(Q(k)) = r_l(Q(l)) \). This is identical to case 1.

Case 3: \( |r_k(Q(k))| < |r_l(Q(l))| \)
Part 1: $r_l(Q(l)) \geq 0$

$$r_l(Q(l) \setminus Y \cup X) = \frac{[W(Q(l)) - W(Y) + W(X)]}{p(l)S} - 1$$

$$= -\frac{[W(Y) - W(X)]}{p(l)S} + \frac{W(Q(l))}{p(l)S} - 1$$

$$\geq r_l(Q(l))$$

$$r_l(Q(l) \setminus Y \cup X) \geq r_l(Q(l)) \geq 0$$

$$|r_l(Q(l) \setminus Y \cup X)| \geq |r_l(Q(l))|$$

Therefore $\max(|r_k(Q(k) \setminus X \cup Y)|, |r_l(Q(l) \setminus Y \cup X)|) \geq \max(|r_k(Q(k))|, |r_l(Q(l))|)

Part 2: $r_l(Q(l)) \leq 0$

Since $r_l(Q(l)) \leq r_k(Q(k)) \leq 0$ and $|r_k(Q(k))| < |r_l(Q(l))|$, it must hold that $r_k(Q(k)) = r_l(Q(l))$. This is identical to case 1. 

The above proof states that moving weight from a less loaded processor to a more loaded processor will only make the error worse. While this may seem obvious, it should be noted that this does not hold for the sum of absolute errors metric, which will try to move all of the overloaded weight to the fastest processor. In Theorem 3.3.7 it was simply assumed that weight was being moved from processor $k$ to processor $l$. For metric $R$ it is assumed that $k$ is less loaded than $l$. Lemma 3.3.17 however, states that processor $k$ being less loaded than processor $l$ implies that weight is always being moved to the less loaded processor which is processor $k$.

The following Lemma shows that $r_l(Q(l)) > r_k(Q(k))$ implies that $W(Q(l))\frac{r(l)}{p(l)} - W(Q(k))$ is positive. This value occurs in the criteria in Theorem 3.3.21, and Lemma 3.3.18 will be referenced in the proof of the theorem.
Lemma 3.3.18. For any pair of processors $k$ and $l$ in a system and any two sets of tasks $Y$ and $X$ such that $Y \subseteq Q(l)$ and $X \subseteq Q(k)$, if $r_k(Q(k)) < r_l(Q(l))$ then

$$W(Q(l))\frac{p(l)}{p(l)} - W(Q(k)) > 0.$$ 

Proof. of Lemma 3.3.18

\[
\begin{align*}
    r_l(Q(l)) &> r_k(Q(k)) \\
    \text{From definition of } r_n(Y) \\
    \frac{W(Q(l))p(l)}{p(l)S} - 1 &> \frac{W(Q(k))p(k)}{p(k)S} - 1 \\
    \frac{W(Q(l))p(l)}{p(l)S} &> \frac{W(Q(k))p(k)}{p(k)S} \\
    \frac{W(Q(l))}{p(l)} &> \frac{W(Q(k))}{p(k)} \\
    W(Q(l))\frac{p(k)}{p(l)} &> W(Q(k)) \\
    W(Q(l))\frac{p(k)}{p(l)} - W(Q(k)) &> 0
\end{align*}
\]

\[\square\]

Lemma 3.3.19. For any pair of processors $k$ and $l$ in a system and any two sets of tasks $Y$ and $X$ such that $Y \subseteq Q(l)$ and $X \subseteq Q(k)$ and $r_k(Q(k)) < r_l(Q(l))$ and $W(Y) - W(X) > 0$, if $r_l(Q(l)) \geq 0$ and $r_l(Q(l) \setminus Y \cup X) \geq 0$ then $|r_l(Q(l) \setminus Y \cup X)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)$. 
Proof. of Lemma 3.3.19 To show that \( |r_l(Q(l) \setminus Y \cup X)| < \max(|r_k(Q(k))|, |r_l(Q(l))|) \) holds, consider how \( |r_l(Q(l) \setminus Y \cup X)| \) compares to \( |r_l(Q(l))| \).

\[
    r_l(Q(l) \setminus Y \cup X) = \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1
    = \frac{-[W(Y) - W(X)]P}{p(l)S} + \frac{W(Q(l))P}{p(l)S} - 1
\]

Since \( W(Y) - W(X) > 0 \) the first term is negative.

\( < 0 + r_l(Q(l)) \)

From the assumptions in the Lemma \( r_l(Q(l) \setminus Y \cup X) \geq 0 \).

\[
    0 \leq r_l(Q(l) \setminus Y \cup X) < r_l(Q(l))
    \]

\( |r_l(Q(l) \setminus Y \cup X)| < |r_l(Q(l))| \)

\( |r_l(Q(l) \setminus Y \cup X)| < \max(|r_k(Q(k))|, |r_l(Q(l))|) \)

\( \Box \)

Lemma 3.3.19 essentially states that if processor \( l \), the more loaded processor, was overloaded before the move and it is still overloaded after the move that the error contributed by processor \( l \) will by reduced. This will be used in Lemma 3.3.21 to simplify cases where \( l \) is overloaded and the load balancer is not moving enough weight to fix \( l \)’s overload.

**Lemma 3.3.20.** For any pair of processors \( k \) and \( l \) in a system and any two sets of tasks \( Y \) and \( X \) such that \( Y \subseteq Q(l) \) and \( X \subseteq Q(k) \) and \( r_k(Q(k)) < r_l(Q(l)) \) and \( W(Y) > W(X) \), if \( r_k(Q(k)) \leq 0 \) and \( r_k(Q(k) \setminus X \cup Y) \leq 0 \) then \( |r_k(Q(k) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|) \).
Proof. of Lemma 3.3.20 To show that \(|r_k(Q(k) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)\) holds, consider how \(|r_k(Q(k) \setminus X \cup Y)|\) compares to \(|r_k(Q(k))|\).

\[
r_k(Q(k) \setminus X \cup Y) = \left[\frac{W(Q(k)) + W(Y) - W(X)}{p(k)S}\right] - 1
= \left[\frac{W(Y) - W(X)}{p(k)S}\right] + \frac{W(Q(k))p(k)S}{p(k)S} - 1
\]

Since \(W(Y) - W(X) > 0\) the first term is positive.

\[> 0 + r_k(Q(k))\]

From the assumptions in the Lemma \(r_k(Q(k) \setminus X \cup Y) \leq 0\).

\[0 \geq r_k(Q(k) \setminus X \cup Y) > r_k(Q(k))\]

\[|r_k(Q(k) \setminus X \cup Y)| < |r_k(Q(k))|\]

\[|r_k(Q(k) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)\]

Lemma 3.3.20 is similar to Lemma 3.3.19, but for the under loaded processor, processor \(k\). If processor \(k\) is under loaded before the load balancer makes a move and it is still under loaded after the move then the error contribution due to processor \(k\) will be reduced. This will be used in Lemma 3.3.21 to simplify cases where \(k\) is under loaded and the load balancer is not moving enough weight to fix the under load on processor \(k\).

**Theorem 3.3.21.** For any pair of processors \(k\) and \(l\) in a system and the sets of tasks \(Y\) and \(X\) such that \(Y \subseteq Q(l)\) and \(X \subseteq Q(k)\) and \(r_k(Q(k)) < r_l(Q(l))\) and \(W(Y) > W(X)\), \(\max(|r_k(Q(k) \setminus X \cup Y)|, |r_l(Q(l) \setminus Y \cup X)|) < \max(|r_k(Q(k))|, |r_l(Q(l))|)\) is true if and only if one of the following criteria is true:

1. \(r_k(Q(k)) \leq 0\) and \(r_l(Q(l)) > 0\) and \(r_k(Q(k) \setminus X \cup Y) > 0\) and \(r_l(Q(l) \setminus X \cup Y) > 0\) and \(W(Y) - W(X) < \max\left(W(Q(l))\frac{p(k)}{p(l)} - W(Q(k)), 2(\alpha_k S - W(Q(k)))\right)\)
2. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) \geq 0 \) and \( r_k(Q(k) \setminus X \cup Y) \leq 0 \) and \( r_l(Q(l) \setminus X \cup Y) > 0 \)

3. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus X \cup Y) \leq 0 \) and
\[
W(Y) - W(X) < \max \left( W(Q(l))^{\frac{p(k)}{p(l)}} - W(Q(k)), 2[\alpha_k S - W(Q(k))] \right) \text{ and }
W(Y) - W(X) < \max \left( 2W(Q(l)) - 2\alpha_l S, W(Q(l)) - W(Q(k))^{\frac{p(l)}{p(k)}} \right)
\]

4. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus X \cup Y) \leq 0 \) and
\[
W(Y) - W(X) < \max \left( 2W(Q(l)) - 2\alpha_l S, W(Q(l)) - W(Q(k))^{\frac{p(l)}{p(k)}} \right)
\]

5. \( r_k(Q(k)) > 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus X \cup Y) > 0 \) and
\[
W(Y) - W(X) < W(Q(l))^{\frac{p(k)}{p(l)}} - W(Q(k))
\]

6. \( r_k(Q(k)) > 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus X \cup Y) \leq 0 \) and
\[
W(Y) - W(X) < 2W(Q(l)) - 2\alpha_l S \text{ and } W(Y) - W(X) < W(Q(l))^{\frac{p(k)}{p(l)}} - W(Q(k))
\]

7. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) \leq 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus X \cup Y) \leq 0 \) and
\[
W(Y) - W(X) < W(Q(l)) - W(Q(k))^{\frac{p(l)}{p(k)}} \text{ and } W(Y) - W(X) < 2\alpha_k S - 2W(Q(k))
\]

8. \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) \leq 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus X \cup Y) \leq 0 \) and
\[
W(Y) - W(X) < W(Q(l)) - W(Q(k))^{\frac{p(l)}{p(k)}}
\]

k proof of Theorem 3.3.21 is provided in appendix B.

**Theorem 3.3.22.** The error \( R \) of a system will be reduced or unchanged if the error \( \max(|r_k(Q(k))|, |r_l(Q(l))|) \) is reduced for some pair of processors \( k \) and \( l \).

**Proof.** of Theorem 3.3.22 Let \( R \) be the error of the system before the move is made then \( R = \max_{n \in N}(|r_n(Q(n))|) \) where \( N \) is the set of all processor in the system.

Let \( k \) and \( l \) be 2 processors from the set \( N \). The theorem assumes that there exist a move that will reduce the error of the pair \( k, l \). Let the sets of tasks \( Y \) and \( Q \) represent that move such that \( Y \subseteq Q(l) \) is the set of tasks moved from \( l \) to \( k \) and \( X \subseteq Q(k) \) is the set of tasks moved from processor \( k \) to \( l \). Then
\[ \max(|r_k(Q(k) \setminus X \cup Y)|, |r_i(Q(l) \setminus Y \cup X)|) < \max(|r_k(Q(k))|, |r_i(Q(l))|) \] by Lemma 3.3.21.

Now consider the error \( R \) as the error of 2 subsystems one containing the processors \( k \) and \( l \), while the other subsystem contains all of the other processors.

Then \( R = \max(\max_{n \in N \setminus k,l}(|r_n(Q(n))|), \max_{n \in k,l}(|r_n(Q(n))|)) \). Let \( R' \) be the error after the move is made, thus

\[ R' = \max(\max_{n \in N \setminus k,l}(|r_n(Q(n))|), \max(|r_k(Q(k) \setminus X \cup Y)|, |r_i(Q(l) \setminus Y \cup X)|)) \].

Since \( \max(|r_k(Q(k) \setminus X \cup Y)|, |r_i(Q(l) \setminus Y \cup X)|) < \max_{n \in k,l}(|r_n(Q(n))|) \), \( R' \leq R \). Therefore, a move that reduce the error of a pair of processors will not increase the error of the system containing that pair. \( \square \)

### 3.3.9 Simplifying the Criteria for \( R \)

The criteria for the maximum absolute relative error, \( R \), is in many ways simpler than the criteria implied by the \( E \) error metric. First the criteria for \( R \) are as sensitive to the performance of the processors as the criteria for \( E \). In the \( E \) criteria some cases only apply when one processor is faster than the other is; the \( R \) criteria has no such cases. Furthermore, the only performance ratios that occur in the \( R \) criteria are \( \frac{p(k)}{p(l)} \) and \( \frac{p(l)}{p(k)} \), which are simple and straightforward. Since none of the \( R \) criteria are dependent on which processor is faster, the over load/under load scenario, \( r_k(Q(k)) \leq 0 \land r_i(Q(l)) > 0 \), has only 4 criteria. The following substitutions will be used to simplify the \( R \) criteria:

\[
\begin{align*}
\delta &= W(Y) - W(X) \\
\sigma_k &= W(Q(k)) - \alpha_k S \\
\sigma_i &= W(Q(l)) - \alpha_i S
\end{align*}
\]

\( \sigma_k \) is the amount of weight by which processor \( k \) is over loaded. If processor \( k \) is underloaded it will be negative. \( \sigma_i \) is the amount of weight by which processor \( l \) is
overloaded. If processor \( l \) is under loaded \( \sigma_l \) will be negative. The equivalencies
3.24, 3.25, 3.26, 3.27, 3.28, 3.29 will also be used to remove references to the end
state from the criteria. The criteria form Theorem 3.3.21 can then be rewritten as:

1. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \land \delta > -\sigma_k \land \delta < \sigma_l \land \delta < \max\left( W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k \right) \)

2. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \land \delta \leq -\sigma_k \land \delta < \sigma_l \)

3. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \land \delta > -\sigma_k \land \delta \geq \sigma_l \land \delta < \max\left( W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k \right) \land \delta < \max\left( 2\sigma_l, W(Q(l)) - W(Q(k)) \right) \)

4. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \land \delta \leq -\sigma_k \land \delta \geq \sigma_l \land \delta < \max\left( 2\sigma_l, W(Q(l)) - W(Q(k)) \right) \)

5. \( r_k(Q(k)) > 0 \land r_l(Q(l)) > 0 \land \delta < \sigma_l \land \delta < W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)) \)

6. \( r_k(Q(k)) > 0 \land r_l(Q(l)) > 0 \land \delta \geq \sigma_l \land \delta < 2\sigma_l \land \delta < W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)) \)

7. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) \leq 0 \land \delta > -\sigma_k \land \delta < W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \land \delta < -2\sigma_k \)

8. \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) \leq 0 \land \delta \leq -\sigma_k \land \delta < W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \)

3.3.9.1 Underload/Overload

The under load/over load scenario describes situations in which processor \( k \) is
underloaded and processor \( l \) is overloaded. Under load/Over load describe the
criteria where \( r_k(Q(k)) \leq 0 \land r_l(Q(l)) > 0 \) is true, which is the first four criteria. To
simplify the under load/over load criteria four cases will be considered. First
which processor has more error by weight, \(-\sigma_k < \sigma_l\) and \(-\sigma_k \geq \sigma_l\). Then in each of
those cases the processors the greater error will be compared to one of the ‘max’
values from the criteria.
Case 1: $-\sigma_k < \sigma_l$

The assumption $-\sigma_k < \sigma_l$ does not change criteria 1. From criteria 2 the statement
\[ \delta < \sigma_l \]
remains unchanged, since this implies $-\sigma_k < \sigma_l \land \delta \leq -\sigma_k$. Similarly the statement $\delta > -\sigma_k$ is removed from criteria 3, since $-\sigma_k < \sigma_l \land \delta \geq \sigma_l \implies \delta > -\sigma_k$.

Criteria 4 is false, since $\delta \leq -\sigma_k \land \delta \geq \sigma_l \implies -\sigma_k \geq \sigma_l$ which violates the assumption. The Criteria are now:

1. $-\sigma_k < \delta < \sigma_l \land \delta < \max\left(W(Q(l))\frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right)$

2. $\delta \leq -\sigma_k$

3. $\delta \geq \sigma_l \land \delta < \max\left(W(Q(l))\frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right) \land \delta < \max\left(2\sigma_l, W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}\right)$

4. false

Case 1.1: $-\sigma_k < \sigma_l$ and $\sigma_l \leq \max\left(w(Q(l))\frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right)$

Criteria 1 is the only criteria changed by this assumption. The statement
\[ \delta < \max\left(W(Q(l))\frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right) \]
is removed, since $\sigma_l \leq \max\left(w(Q(l))\frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right) \land \delta < \sigma_l$ imply that it is true. The criteria now become:

1. $-\sigma_k < \delta < \sigma_l$

2. $\delta \leq -\sigma_k$

3. $\delta \geq \sigma_l \land \delta < \min\left(\max\left(W(Q(l))\frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right), \max\left(2\sigma_l, W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}\right)\right)$

4. false
When combined these criteria yield the following:

\[
\delta \leq -\sigma_k \lor -\sigma_k < \delta < \sigma_l \lor \delta \geq \sigma_l \land \delta < \max\left(\frac{W(Q(l))}{p(l)} - W(Q(k)), -2\sigma_k\right)
\]

\[
\land \delta < \max\left(2\sigma_l, W(Q(l)) - W(Q(k))\right) \frac{p(l)}{p(k)}
\]

\[
\delta < \max\left(W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right) \land \delta < \max\left(2\sigma_l, W(Q(l)) - W(Q(k))\right) \frac{p(l)}{p(k)}
\]

\[
\delta < \min\left(\max\left(W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right), \max\left(2\sigma_l, W(Q(l)) - W(Q(k))\right)\right)
\]

(3.30)

**Case 1.2:** $-\sigma_k < \sigma_l$ and $\sigma_l > \max\left(W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right)$

The assumptions of case 1.2 alter the criteria 1 and 3. Like case 1.1 the assumption of this case decide which value in the function $\max\left(W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right)$ is greater, thus simplifying criteria 1. The assumption also makes criteria 3 false, since it assumes $\sigma_l > \max\left(W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right)$ but case 3 requires $\delta \geq \sigma_l \land \delta < \max\left(W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right)$. Thus with this assumption, the first four criteria reduce to the following:

1. $-\sigma_k < \delta < \max\left(W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right)$.
2. $\delta \leq -\sigma_k$.
3. false.
4. false.

Those criteria, when combined by logical ‘or’ yield:

\[
\delta \leq -\sigma_k \lor -\sigma_k < \delta < \max\left(W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right)
\]

\[
\delta < \max\left(W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k\right)
\]
Now notice that \( \max \left( W(Q(l)) \frac{p(l)}{p(k)} - W(Q(k)), -2\sigma_k \right) \) is the minimum of \( \max \left( W(Q(l)) \frac{p(l)}{p(k)} - W(Q(k)), -2\sigma_k \right) \) and \( \max \left( 2\sigma_l, W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \right) \).

\[
\max \left( 2\sigma_l, W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \right) \\
\geq 2\sigma_l \\
\geq \sigma_l \\
> \max \left( W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_l \right)
\]

Thus in case 1.2 the criteria can be stated as, which is identical to 3.30 from case 1.1. Therefore all of case 1 can be summarized by statement 3.30.

Case 2: \(-\sigma_k \geq \sigma_l\)

Case 2 is almost identical to case 1. The assumption that \(-\sigma_k \geq \sigma_l\) is true, alters the criteria 1, 2, and 3, while criteria 4 remain unchanged. Criteria 1 is false, since \(-\sigma_k < \delta < \sigma_l\) is false. Criteria 2 is simplified to \(\delta < \sigma_l\), since 

\[
\delta < \sigma_l \land -\sigma_k \geq \sigma_l \implies \delta \leq -\sigma_k.
\]

From criteria 3 the statement \(\delta \geq \sigma_l\) is dropped, since \(\delta > -\sigma_k \land -\sigma_k \geq \sigma_l \implies \delta \geq \sigma_l\). The resulting criteria are as follows:

1. false.

2. \(\delta < \sigma_l\).

3. \(\delta > -\sigma_k \land \delta < \max \left( W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k \right) \land \max \left( W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right)\).

4. \(\sigma_l \leq \delta \leq \min \left( -\sigma_k, \max \left( w(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \right) \right)\).

Case 2.1: \(-\sigma_k \geq \sigma_l \land -\sigma_k < \max \left( w(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right)\) the only criteria altered by the additional assumption is criteria 4. The new assumptions decides the result of the function 

\[
\min \left( -\sigma_k, \max \left( w(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \right) \right)\].

The criteria are simplified to:
1. false.

2. $\delta < \sigma_l$.

3. $\delta > -\sigma_k \land 0 < \min \left( W(Q(l)) \frac{p(l)}{p(l)} - W(Q(k)), -2\sigma_k \right) \land \max \left( W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right)$

4. $\sigma_l \leq \delta \leq -\sigma_k$.

The simplified criteria can be combined by logical 'or', with the following result:

$$\delta < \sigma_l \lor \sigma_l \leq \delta \leq -\sigma_k \lor \delta > -\sigma_k \land 0 < \min \left( W(Q(l)) \frac{p(l)}{p(l)} - W(Q(k)), -2\sigma_k \right) \land \max \left( W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right)$$

$$\delta < \max \left( W(Q(l)) \frac{p(l)}{p(l)} - W(Q(k)), -2\sigma_k \right) \land \max \left( W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right)$$

$$\delta < \min \left( \max \left( W(Q(l)) \frac{p(l)}{p(l)} - W(Q(k)), -2\sigma_k \right), \max \left( W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right) \right)$$

(3.31)

**Case 2.2:** $-\sigma_k \geq \sigma_l \land -\sigma_k \geq \max \left( W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right)$

The additional assumption of case 2.2 decides the value of the min function in criteria 4. The new assumption also causes criteria 3 to become false, since $-\sigma_k \geq \max \left( W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right)$ implies that $-\sigma_k < \delta < \max \left( W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right)$ is false. The criteria become the following:

1. false.

2. $\delta < \sigma_l$.

3. false.

4. $\sigma_l \leq \delta \leq \max \left( W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right)$.
These criteria can be combined by logical ‘or’ as follows:

\[ \delta < \sigma_l \lor \sigma_l \leq \delta \leq \max \left( W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right) \]

\[ \delta \leq \max \left( w(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right) \]

Now notice that \( \max \left( w(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right) \) is less than \( \max \left( W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k \right) \), both of which occur in statement 3.31.

\[ \max \left( W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), -2\sigma_k \right) \geq -2\sigma_k \]
\[ \geq -\sigma_k \]
\[ \geq \max \left( w(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right) \]

Thus the result of the min function in statement 3.31 is equal to \( \max \left( w(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, 2\sigma_l \right) \) in case 2.2. Therefore all of case 2 can be summarized by the statement 3.31. Furthermore, 3.31 and 3.30 are the same statement, thus both cases, the entire over loaded/under loaded scenario can be describe by either 3.31 or 3.30.

### 3.3.9.2 Overload/Overload

This scenario occurs when both processor are overloaded, which is represented by the statement \( r_k(Q(k)) > 0 \land r_l(Q(l)) > 0 \). Criteria 5 and 6 are the overload/overload criteria. These criteria can be simplified into a single statement without separate cases. The criteria are as follows:

5. \( \delta < \sigma_l \land \delta < W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)) \).

6. \( \delta \geq \sigma_l \land \delta < \min \left( 2\sigma_l, W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)) \right) \).
Since \( \delta < \sigma_l \implies \delta < 2\sigma_l \) criteria 5 can be rewritten as follows:

\[
\delta < \sigma_l \land \delta < 2\sigma_l \land \delta < W(Q(l))\frac{p(k)}{p(l)} - W(Q(k))
\]

\[
\delta < \sigma_l \land \delta < \min\left(2\sigma_l, W(Q(l))\frac{p(k)}{p(l)} - W(Q(k))\right)
\]

Now the criteria become:

5. \( \delta < \sigma_l \land \delta < \min\left(2\sigma_l, W(Q(l))\frac{p(k)}{p(l)} - W(Q(k))\right) \).

6. \( \delta \geq \sigma_l \land \delta < \min\left(2\sigma_l, W(Q(l))\frac{p(k)}{p(l)} - W(Q(k))\right) \).

The only difference between the two criteria 5 and 6 are the statements \( \delta < \sigma_l \) and \( \delta \geq \sigma_l \), one of which is always true. Thus, the following summarize overload/overload scenario:

\[
\delta < \min\left(2\sigma_l, W(Q(l))\frac{p(k)}{p(l)} - W(Q(k))\right)
\]

This function can be rewritten in another interesting way. First realize that in the overload/overload case \( \sigma_l > W(l) - W(k)\frac{p(l)}{p(k)} \).

\[
\sigma_l = W(Q(l)) - \alpha_l S
\]

\[
= W(Q(l)) - \frac{p(l)}{P} S
\]

\[
= W(Q(l)) - \frac{p(k)}{P} \frac{p(l)}{p(k)} S
\]

\[
= W(Q(l)) - \alpha_k S \frac{p(l)}{p(k)}
\]

Note: \( r_k(Q(k)) > 0 \implies W(Q(k)) > \alpha_k S \)

\[
\sigma_l > W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}
\]

\[
\because 2\sigma_l = \max\left(2\sigma_l, W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}\right)
\]
Second, realize that $-\sigma_k < W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k))$ in the overload/overload case.

$$-\sigma_k = \alpha_k - W(Q(k))$$

$$= \frac{p(k)}{p} S - W(Q(k))$$

$$= \frac{p(l) p(k)}{p} S - W(Q(k))$$

$$= \alpha_l S \frac{p(k)}{p} S - W(Q(k))$$

Note: $r_i(Q(l)) > 0 \implies W(Q(l)) > \alpha_l S$

$$-\sigma_k < W(l) \frac{p(k)}{p(l)} - W(Q(k))$$

Note: $-\sigma_k < 0$

$$-2\sigma_k < W(l) \frac{p(k)}{p(l)} - W(Q(k))$$

.: $W(l) \frac{p(k)}{p(l)} - W(Q(k)) = \max \left( -2\sigma_k, W(l) \frac{p(k)}{p(l)} - W(Q(k)) \right)$

With those two inequalities overload/overload the criteria can be rewritten as:

$$\delta < \min \left( \max \left( 2\sigma_l, W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \right), \max \left( -2\sigma_k, W(l) \frac{p(k)}{p(l)} - W(Q(k)) \right) \right)$$

(3.32)

3.3.9.3 Underload/Underload

The under load/under load scenario occurs when both processor are under loaded, which corresponds to the statement $r_k(Q(k)) \leq 0 \land r_l(Q(l)) \leq 0$. This is true for the last two criteria 7 and 8. The scenario is similar to the over load/over load scenario. The criteria for under load/under load are as follows:

7. $\delta > -\sigma_k \land \delta < W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \land \delta < -2\sigma_k$.

8. $\delta \leq -\sigma_k \land \delta < W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}$. 
In criteria 8 $\delta \leq -\sigma_k$ implies that $\delta < -2\sigma_k$. Note that if $-\sigma_k = 0$ then this criteria will always be false. Thus criteria 8 can be rewritten as follows:

$$\delta \leq -\sigma_k \land \delta < W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \land \delta < -2\sigma_k$$

$$\delta \leq -\sigma_k \land \delta < \min\left(W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, -2\sigma_k\right)$$

The criteria can now be written as:

7. $\delta > -\sigma_k \land \delta < \min\left(W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, -2\sigma_k\right)$.

8. $\delta \leq -\sigma_k \land \delta < \min\left(W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, -2\sigma_k\right)$.

The only difference between criteria 7 and 8 is the statements $\delta > -\sigma_k$ and $\delta \leq -\sigma_k$.

Thus the under load/under the following statement can summarize load scenario:

$$\delta < \min\left(W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}, -2\sigma_k\right)$$

The criteria can be further transformed in a similar manner to that of the overload/overload criteria. The under load/under load case however,

$$\sigma_l \leq W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}.$$
Similarly in the under load/under load scenario $-\sigma_k \geq W(Q(l))\frac{p(k)}{p(l)} - W(Q(k))$.

$$-\sigma_k = \alpha_k - W(Q(k))$$

$$= \frac{p(k)}{P} S - W(Q(k))$$

$$= \frac{p(l) p(k)}{p(l)} S - W(Q(k))$$

$$= \alpha_l S \frac{p(k)}{p(l)} S - W(Q(k))$$

Note: $r_l(Q(l)) \leq 0 \Rightarrow W(Q(l)) \leq \alpha_l S$

$-\sigma_k \geq W(l)\frac{p(k)}{p(l)} - W(Q(k))$

Note: $-\sigma_k < 0$

$-2\sigma_k \geq W(l)\frac{p(k)}{p(l)} - W(Q(k))$

$\therefore -2\sigma_k = \max \left( -2\sigma_k, W(l)\frac{p(k)}{p(l)} - W(Q(k)) \right)$

With those two inequalities the under load/under load criteria can be rewritten as:

$$\delta < \min \left( \max \left( 2\sigma_l, W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)} \right), \max \left( -2\sigma_k, W(l)\frac{p(k)}{p(l)} - W(Q(k)) \right) \right)$$

(3.33)

Finally notice that statement 3.30, 3.31, 3.32, and 3.33 are all the same. Thus the entire $R$, metric regardless load or processor performance can is described by one inequality.

### 3.3.9.4 Understanding the One Criteria

Each part of the one criteria that describes the $R$ metric has a reasonably intuitive meaning. The goal of the $R$ metric is minimize the maximum absolute value of the relative error, by doing so for each pair. The maximum amount of weight that can be moved from one processor to another is thus the minimum of pair of values each corresponding to one of the processors, hence the min function.
in 3.33. The two value in the min, each representing one of the processors, are the result of max functions. The max function for processor $l$ is:

$$\max\left(2\sigma_l, W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}\right)$$

The first term $2\sigma_l$ the amount of weight that can be moved away from $l$ before relative error becomes the negative of its current value. If $l$ is under loaded this is a negative amount. The second term, $W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}$, is the amount that be moved away from $l$ before its relative error will be equal to that of $k$. The max function is used, since the metric is only concerned with the maximum error, so the error on $l$ may increase if $k$ previously had more error. The max function for processor $k$ is:

$$\max\left(-2\sigma_k, W(l)\frac{p(k)}{p(l)} - W(Q(k))\right)$$

This is entirely similar to the function for processor $l$. The term $-2\sigma_k$ is the amount of weight that can be moved to $k$ before it exceeds it prior error. Likewise $W(l)\frac{p(k)}{p(l)} - W(Q(k))$ is the amount of weight that can be moved to $k$ before it exceeds the prior error on processor $l$.

### 3.3.10 Optimal Choice of Weight for Metric R

**Lemma 3.3.23.** Given a pair of processors, $k$ and $l$, if $r_k(Q(k)) = r_l(Q(l))$, then moving any amount of weight from processor $l$ to processor $k$ will not decrease metric $R$.

**Proof.** Proof of Lemma 3.3.23 Assume that $k$ and $l$ are processors such that $r_k(Q(k)) = r_l(Q(l))$. Let $Y$ and $X$ be sets of tasks such that $Y \subseteq Q(l)$ and $X \subseteq Q(k)$, then $\delta = W(Y) - W(X)$. Assume with out loss of generality that $\delta > 0$.

It is demonstrated that moving $\delta$ weight from processor $l$ to $k$ will increase the maximum error of the pair in two case $r_k(Q(k)) = r_l(Q(l)) > 0$ and $r_k(Q(k)) = r_l(Q(l)) < 0$. 
Case 1: \( r_k(Q(k)) = r_l(Q(l)) > 0 \)

\[
r_k(Q(k) \setminus X \cup Y) = \frac{W(Q(k)) - W(X) + W(Y)}{p(k)S} - 1
\]

\[
= \frac{W(Q(k)) + \delta}{p(k)S} - 1
\]

\[
= \frac{W(Q(k))}{p(k)S} + \frac{\delta}{p(k)S} - 1
\]

\[
= r_k(Q(k)) + \frac{\delta}{p(k)S}
\]

Since \( \delta > 0 \).

\( r_k(Q(k) \setminus X \cup Y) > r_k(Q(k)) \)

Since \( r_k(Q(k)) > 0 \)

\[|r_k(Q(k) \setminus X \cup Y)| > |r_k(Q(k))|\]

Since \( r_k(Q(k)) = r_l(Q(l)) \)

\[
\max(|r_k(Q(k) \setminus X \cup Y)|, |r_l(Q(l) \setminus Y \cup X)|) > \max(|r_k(Q(k))|, |r_l(Q(l))|)
\]

\( \therefore \) the maximum error of the pair is increased and the error of the system \( R \) is not decreased.
Case 1: $r_k(Q(k)) = r_l(Q(l)) < 0$

\[
\begin{align*}
r_l(Q(l) \setminus Y \cup X) &= \frac{W(Q(l)) - W(Y) + W(X)}{p(l)S} - 1 \\
&= \frac{W(Q(l)) - \delta}{p(l)S} - 1 \\
&= \frac{W(Q(l))}{p(l)S} - \frac{\delta}{p(l)S} - 1 \\
&= r_l(Q(l)) - \frac{\delta}{p(l)S}
\end{align*}
\]

Since $\delta > 0$.

\[
r_l(Q(l) \setminus Y \cup X) < r_l(Q(l))
\]

Since $r_l(Q(l)) < 0$

\[
|r_l(Q(l) \setminus Y \cup X)| > |r_l(Q(l))|
\]

Since $r_k(Q(k)) = r_l(Q(l))$

\[
\max(|r_k(Q(k) \setminus X \cup Y)|, |r_l(Q(l) \setminus Y \cup X)|) > \max(|r_k(Q(k))|, |r_l(Q(l))|)
\]

\[\because\] the maximum error of the pair is increased and the error of the system $R$ is not decreased. \quad \square

**Theorem 3.3.24.** Given a pair of processors, $k$ and $l$, such that $r_k(Q(k)) \leq r_l(Q(l))$, the minimum value of metric $R$ is achieved when $\delta^*_R = \frac{p(l)W(l) - p(k)W(k)}{p(l) + p(k)}$ is moved from processor $l$ to $k$.

**Proof.** Proof of Theorem 3.3.24 Let $k$ and $l$ be a pair of processors such that $r_k(Q(k)) \leq r_l(Q(l))$. Let $\delta^*_R = \frac{p(l)W(l) - p(k)W(k)}{p(l) + p(k)}$, and $Y$ and $X$ be sets of tasks such that $Y \subseteq Q(l)$ and $X \subseteq Q(k)$ and $W(Y) - W(X) = \delta^*_R$. 
The first step towards showing that $\delta^*_R$ is an optimal choice, is to demonstrate that $r_k(Q(k) \setminus X \cup Y) = r_l(Q(l) \setminus Y \cup X)$.

\[
r_k(Q(k) \setminus X \cup Y) = \frac{[W(Q(k)) - W(X) + W(Y)]P}{p(k)S} - 1
\]

\[
= \frac{[W(Q(k)) + \delta^*_R]P}{p(k)S} - 1
\]

\[
= \left( \frac{W(Q(k)) + \delta^*_R}{p(k)} \right) \left( \frac{P}{S} \right) - 1
\]

\[
= \left( \frac{W(Q(k))}{p(k)} + \frac{p(k)W(l) - p(l)W(k)}{p(l) + p(k)} \left( \frac{1}{p(k)} \right) \left( \frac{P}{S} \right) - 1
\]

\[
= \left( \frac{p(l)W(Q(k)) + p(k)W(Q(k))}{(p(l) + p(k)p(k)} + \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{(p(l) + p(k)p(k)} \right) \left( \frac{P}{S} \right) - 1
\]

\[
= \left( \frac{W(Q(k)) + p(k)W(Q(l))}{(p(l) + p(k)} \right) \left( \frac{P}{S} \right) - 1
\]

\[
= \left( \frac{p(l)W(Q(k)) + p(l)W(Q(l))}{p(l) + p(k)p(l)} \right) \left( \frac{P}{S} \right) - 1
\]

\[
= \left( \frac{p(l)W(Q(l)) + p(k)W(Q(l))}{(p(l) + p(k)p(l)} + \frac{p(l)W(Q(k)) - p(k)W(Q(l))}{(p(l) + p(k)p(l)} \right) \left( \frac{P}{S} \right) - 1
\]

\[
= \left( \frac{W(Q(l))}{p(l)} - \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{(p(l) + p(k)} \left( \frac{1}{p(l)} \right) \left( \frac{P}{S} \right) - 1
\]

\[
= \frac{W(Q(l))P}{p(l)S} - \frac{\delta^*_R P}{p(l)S} - 1
\]

\[
= \frac{[W(Q(l)) - \delta^*_R]P}{p(l)S} - 1
\]

\[
r_k(Q(k) \setminus X \cup Y) = r_l(Q(l) \setminus Y \cup X)
\]

Since $\delta^*_R$ make the error on both processors equal, by lemma 3.3.23, $\delta^*_R$ must be an optimal choice of weights to move to processor $k$ from $l$. 

\[\square\]
3.4 Pair Balancing

In addition to minimizing one of the error metrics $E$ or $R$, the load balancer can also be requested to balance the system as pairs of processors. If the load balancer is balancing the system as pairs it will try to pair of processors if they are the entire system. Thus the total weight is only the weight of the tasks on the selected pair and the total performance is the performance of the selected pair. The load balancer then minimizes the difference in the load of the two processors in the selected pair.

Pair balancing is simpler and more scalable than minimizing the system error. Minimizing the system error requires calculating the error metric or following the rules outlines in section 3.3. Both of those methods however, require knowing the total weight of the system and the total performance. These system totals require information from each processor that may change frequently. These totals can be expensive to calculate on large systems. Pair balancing however, does not need these totals and can be performed without any information from the rest of the system. This method still searches the entire system for the most loaded processor, but if this became to costly the selection method could be modified to only search a subset of the system.

The disadvantage of pair balancing is that it could make the total system error worse. In section 3.3, points were made where cases did not apply in the two processor case. As such pair balancing will ignore these cases and may violate the rules they imply. The following examples demonstrate how pair balancing can increase the total system error.

Example 3.9. In this example pair balancing will improve the balance of the pair $k$ and $l$ by reducing the difference in their load, while increasing the error metric $E$. Consider a
system with 3 processors k, l, and j. Let processor k have a total weight of 3 and a
performance of 4, processors l has a total weight of 2 and a performance of 6, j has a total
weight of 10 and a performance of 15. Thus the system has a total weight of $S = 20$ and a
total performance of $P = 20$ and the processors have the following load, $\frac{W(Q(n))}{P(n)}$ and error

$$|r_n(Q(n))| = |\frac{W(Q(n))P}{P(n)S} - 1|.$$ 

Then $0.25 + 0.6667 + 0.5 = 1.4167$ and the difference in the load between processors k and l
is $0.75 - 0.3333 = 0.417$. Now assume the pair balancer moves a task of weight 1 from
processor k to l, then the system becomes.

<table>
<thead>
<tr>
<th></th>
<th>k</th>
<th>l</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>4</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>Weight</td>
<td>3</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>Load</td>
<td>0.75</td>
<td>0.3333</td>
<td>1.5</td>
</tr>
<tr>
<td>$r_n(Q(n))$</td>
<td>0.25</td>
<td>0.6667</td>
<td>0.5</td>
</tr>
</tbody>
</table>

The pair of processors k and l is perfectly balanced, but the system error metric is $E$ has increased to
$0.5 + 0.5 + 0.5 = 1.5$.

**Example 3.10.** This example demonstrates that pair balancing can also increase the error
metric $R$. Assume the system has 3 processors k, l, and j, with performances of 20, 80, and
300 respectively, and total weights of 20, 40, and 340. Then the system has a total weight

$$\begin{array}{|c|c|c|c|}
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>20</td>
<td>80</td>
<td>300</td>
</tr>
<tr>
<td>Weight</td>
<td>20</td>
<td>40</td>
<td>340</td>
</tr>
<tr>
<td>Load</td>
<td>1</td>
<td>0.5</td>
<td>1.1333</td>
</tr>
<tr>
<td>$r_n(Q(n))$</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>
\end{array}$$ 

of $S = 400$ and a total performance of $P = 400$. The
error metric $R$ is then equal to 0.5 and the pair $k$ and $l$ have a load difference of 0.5. Now assume the pair balancer moves 13 weight from processor $k$ to $l$. Then the system enters the following state.

<table>
<thead>
<tr>
<th></th>
<th>$k$</th>
<th>$l$</th>
<th>$j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>20</td>
<td>80</td>
<td>300</td>
</tr>
<tr>
<td>Weight</td>
<td>7</td>
<td>53</td>
<td>340</td>
</tr>
<tr>
<td>Load</td>
<td>0.35</td>
<td>0.6625</td>
<td>1.1333</td>
</tr>
<tr>
<td>$r_n(Q(n))$</td>
<td>0.65</td>
<td>0.3375</td>
<td>0.1333</td>
</tr>
</tbody>
</table>

The error metric $R$ has now increased to 0.65, but the difference in load of the pair $k$ and $l$ has decreased to 0.3125.

### 3.5 Summary

The Design chapter has described an asymmetric proportional share system, and proposed how a load balancer could implemented to help tasks acquire there appropriate share of processing time. In other proportional share system tasks are allocated time on the processor. Since time is more valuable on certain processor than it is on others this model was changed to allocation performance in section 3.3.2. On symmetric multiprocessor systems a task may have a weight too large to be serviced by any processor in the system. In contrast an asymmetric system could contain a task with a weight to great to be serviced by any processor or only some of the processor. This is an infeasible weight assignment and is discussed int section 3.3.3.

The Design chapter also defined tow error metrics for measuring the how well balanced a system is. One error metric, $E$, gives an overall measure of the system, showing in general how the tasks are doing. The other metric, $R$, indicates how much error is being encountered by the task or tasks with the most error. For each of these metrics proofs are provided for a minimum set of rules that any load balancer must be constrained to in order to never increase the error. Since the
problem is in general to difficult to be solved, these constraints provide a method of evaluating if a heuristic will ever make the balance of the system worse.

This section also provides a more scalable method for balancing that may make the system error worse. This method and the direct minimization of the error metrics will be evaluated in the Testing chapter.
4 Testing

4.1 The Test Environment

For this project, a load balancer and a scheduler were implemented in the Linux kernel version 2.6.22. The scheduler is simply single processor implementation of EEVDF [1] which is already well studied. The goal of testing is demonstrate the characteristics of the load balancer and compare it to the already existing CFS load balancer. Since CFS was added to the official Linux kernel after version 2.6.22, kernel version 2.6.26 will be used to test the CFS load balancer.

The tests themselves will be performed on an AMD Phenom 9600 Quad-core. Each core has its own L1 and L2 cache, and it own power state. This allows the system to become asymmetric by running some cores at full speed 2.30GHz and some of them at half speed 1.15GHz. All four cores share L3 cache and RAM, so the system is not NUMA. This means that only the asymmetric properties of the load balancer will be tested. Locality benefits would require separate memory pools to be well tested; as such, nothing specific has been implemented to handle NUMA systems.

The goal of the load balancer is to maintain equal performance to weight ratios on each processor in a dynamic asymmetric multiprocessor system, where tasks enter and leave the system frequently and processor may change performance occasionally. The implemented load balancer has three heuristics it can use to achieve the goal. The first method, referred to as ‘sum’ in the test data, minimizes the sum of the absolute values of the loads on each processor; this is metric E from the design section. The second method minimizes the maximum of the absolute values of the processor loads, this is metric R, and is referred to a ‘max’ in the test data. The last method minimizes the difference in the loads of the
two processors being examined; this method is referred to as 'bal' in the testing data. The third method has the disadvantage that it may make the overall system error worse, but the advantage that it does not require knowledge of the system's state as a whole.

The tests in this section will test how effective the effectiveness of the three heuristics and the CFS at minimize the overall system error. The methods proposed in this paper were implemented in Linux kernel version 2.6.22, which is prior to the introduction of the CFS scheduler. The CFS scheduler was a complete rewrite of the Linux kernel’s scheduler, so there are few differences in behavior.

4.2 Linux Implementation

4.2.1 EEVDF

The first step was to add a Proportional Share scheduler to the Linux kernel. The EEVDF scheduler was chosen for its accuracy and state tracking. Both SFQ and EEVDF have a maximum error of one time slice, but EEVDF keeps track of virtual time as a scale of real time. EEVDF also keep track of start times, stop times, and deadlines for each task, which made tracking errors during implementation easier. SFQ has better run time properties, which would make it a better choice for a production system.

EEVDF was added to the Linux kernel as an additional scheduler type to be run after real-time tasks but before general-purpose tasks. To allow the scheduler to executes general-purpose tasks while proportional share tasks are still available an idle task was created. The idle task is assigned a weight just like any other proportional share task, but when it is scheduled it causes the scheduler to select the next general purpose task. This prevents general-purpose task from being starved by the proportional share tasks.
Another concern when using a PS scheduler in a real system is task creation. When a PS task forks to create a new task it is not clear how to schedule the new task. If the new task is simply create with the same scheduler and weight as it parent then all of the other tasks in the system will some of there runtime. A malicious task could fork many copies of itself to get as much processing time as it wants and starve the rest of the system. Alternatively, some of the parent’s tasks weight could be assigned to the child task such that the total weight of the two tasks is equal to the parent’s previous weight. This would leave other tasks on the system unchanged. However, as the weights reduce a task with weight of one may try to fork. Since the scheduler puts a lot of work into maintaining the correct shares for each task, this implementation assumes that weights of PS task are chosen carefully and should in no way be altered by the scheduler. Thus, all new tasks created by PS tasks are considered general-purpose tasks. Any task that requires PS scheduling must be promoted to proportional share, event children of proportional share tasks. Only a privileged user can promote tasks to be proportional share, since such tasks consume a large amount of the systems resources.

CFS creates all tasks with same scheduler type and weight or priority as there parent. Only privileged users on CFS can increase the weight; however, any task could effectively double its weight by forking. A task that wanted to increases weight could simply make more copies of itself until it had the desired weight.

4.2.2 Load Balancer

The general-purpose load balancer was not removed to create the load balancer proposed in this paper. Instead, the previous load balancer is left to
balance all of the non-PS tasks, and the new load balancer runs immediately after
the old balancer to balance the PS tasks.

The biggest change that had to be made for to the kernel for the new load
balancer was the ability to move the currently running task. To balance the PS
tasks it was sometimes necessary to move the currently running task, which
normally cannot be done. To achieve this, the load balancer sets a flag on the
running task to mark that it should be moved later. Next time the scheduler
selects a new task for that processor to execute; the previously flagged task is
stopped and placed onto its new processor run queue instead of the run queue for
the processor it was previously running on. This method however is error prone
and if a problem is detected, the scheduler cancels the move and simply runs as it
normally would.

The version of the Linux kernel the load balancer was implemented on
always places task that are reentering the system onto the processor they
previously using. Additionally new tasks are place on the same processor as the
task that created them. This leaves all balancing to the load balancer that runs as
part of the scheduler, and differs from the version of the Linux kernel with CFS.
When the scheduler was rewritten for CFS, the kernel was changed so that new
and reentering tasks are always placed on the least loaded processor, which has a
balancing effect itself. This effect can be seen in the test results. Each test is started
with the load balancer disabled, so when the load balancer starts the proposed
methods have a rapid change in the balance. The CFS results do not have this
change since the system was balancing the tasks as the were placed.

The Linux load balancer prior to CFS simply balanced task by the number of
tasks on the processor. CFS essentially has two load balancers. The primary load
balancer runs in interrupt context just like the scheduler and moves the first fit
tasks to resolve imbalances. That is it scans the run queue moving each task if it would improve the balance. When the load balancer runs from process context, however it will move an arbitrary task. While this can make the balance worse, it is usefully for breaking situations in which the balancer is stuck.

4.2.3 Maximum Error

In [7] the subset-sum approximation solutions are measured by a percentage of the optimal solution that the algorithm achieves. Similarly the two metric proposed in this paper can be compared by a percentage of their worst case values.

In order to test and compare the presented load balancing methods it is necessary to know the maximum amount of error that could be achieved in this test environment. The maximum error for both metrics is achieved by placing all tasks on the slowest processor. Since the relative error on a processor is a ratio of the amount of weight on the processor to the amount of weight the processor would ideally have, the slowest processor will have the largest relative error when all of the weight is placed on it. The other processors will all have relative error of -1, since they will be 100% underloaded. The amount of error the slowest processor will have in this scenario can be determined from the performance of the processors.

Assume processor \( l \) is the slowest processor and that all tasks are assigned to \( l \), then \( W(Q(l)) = S \).

\[
r_l(Q(l)) = \frac{W(Q(l))P}{p(l)S} - 1
\]

\[
= \frac{SP}{p(l)S} - 1
\]

\[
= \frac{p}{p(l)} - 1
\]
In a system with only one processor $P = p(l)$, thus $r_l(Q(l)) = 0$. This is the expected result since, a single processor system is always perfectly balanced. In the two processor case $P \geq 2p(l)$, since $l$ is the slowest processor. The maximum possible value for metric $R$ is $R^{\text{MAX}} = r_l(Q(l))$, since processor $l$ has a relative error of at least 1 and all other processor have a relative error of exactly −1. Metric $E$ is the sum of the absolute value of the errors, so it maximum is $E^{\text{MAX}} = R^{\text{MAX}} + |N| − 1$ where $|N|$ is the cardinality of $N$ the set of all processors.

The test environment has a four core processor, so there are 4 processors in each test. Tests 1 and 2 run are asymmetric and one of the processors changes its performance during the test. For the first 30 seconds of tests 1 and 2 the first two cores run at full performance and the second two core run at half performance. During the remainder of tests 1 and 2 the third core increases it performance, while the fourth core remains a half speed. Thus $R^{\text{MAX}}$ for the first half of tests 1 and 2 is $R^{\text{MAX}} \approx \frac{1+1+0.5+0.5}{0.5} - 1 = 5$ and during the second half $R^{\text{MAX}} = \frac{1+1+1+0.5}{0.5} - 1 = 6$. Metric $E$ then has maximum of $E^{\text{MAX}} \approx 5 + 4 - 1 = 8$ during the first half and $E^{\text{MAX}} \approx 6 + 4 - 1 = 9$ during the second half. For tests 3 and 4 all processors run at their full performance for the duration of the test, so $R^{\text{MAX}} \approx \frac{1+1+1+1}{1} - 1 = 3$ and $E^{\text{MAX}} \approx 3 + 4 - 1 = 6$. The values of $R^{\text{MAX}}$ and $E^{\text{MAX}}$ are approximate, because the performance of the processors is measured empirically the exact performance of the cores varies between trials.

### 4.3 Test 1

This test demonstrates the ability of the load balancer to find a minimum error. The test starts by disabling the load balancer, setting cores 0 and 1 to full speed, and setting cores 2 and 3 to half speed. Next, the test creates 80 tasks of various fixed weights. The system is non-deterministic, thus starting with the
load balancer disabled allows the start state to be observed. Five seconds into the
test the load balancer is enable and begins minimizing the error. At thirty seconds
into the test, core 2 switches from half speed to full speed. This demonstrates the
ability of the load balancer to adjust for a simple change. At sixty seconds, the
tasks begin terminating. The tasks used in this repeatedly perform simple math
operation on a block of memory until the sixty seconds has expired. Each task
additionally reports its progress every second. A separate tasks runs as a FIFO
real-time task and polls the state of the load balancer and scheduler every 250ms.
All of the reporting is done using files on a ram to disk to prevent blocking,
however occasionally some of the tasks do block, which causes disturbances in
the data. This test is repeated 300 times and the results are aggregated.

Figure 4.1: Load balancer performance by error metric

The graphs in figure 4.1 show how each balancing method performs under
Metrics E and R.
**Average Error by Metric E and R:** On this graph the average result is shown for each balancing method according to Metric E. ‘Average Error by Metric R’, similarly shows the averages for Metric R. The averages are taken by the mean of all the result during each quarter second period. The error values on the graph for Metric E are significantly larger than those for Metric R due to the fact Metric E is the sum of the errors and Metric is just the maximum of the errors.

On Test 1, the three methods proposed in this paper performed almost identically for both Metrics E and R. The quick descent at second 5 shows that most of the time the load balancer quickly achieves its equilibrium state. These graphs show a small spike just after second 30; this is caused by core 2 doubling its performance, which increases the error. The spike is quickly resolved as the load balancer adjusts to the new performance. The CFS load balancer does not perform as well. This is at least partially because CFS does not make adjustments for the varying performance among processors and cores.

**Maximum Error by Metric E and R:** This pair of graphs shows the maximum error obtained by each of the three methods at any point any time. This is not the worst single run, instead at each point the worst value from any run was found and plotted. This graph primarily shows how bad the metrics can get when the tasks remain static. There are several spikes on this pair of graphs this disturbance is caused by occasionally blocking. The test was setup to prevent blocking, but it still happened occasionally. When a task blocks the processor it was on looses the weight of that task, and when the task becomes active again it weight will be added to the processor that it wakes on. Most of the time the task will resume on the same processor it was on before it blocked. The load balancer attempts to correct changes weight every time it runs, which helps to keep the spikes in error short.
Again, the three proposed methods performed similarly and CFS performed much worse.

**Standard Deviation of Error by Metric E and R:** This graph is the standard deviation at each point in time for each balancing method. The primary purpose of the graph is to show that the results are moderately consistent across trails. The slow decent on this graph, compared Average Error by Metric, show that in some cases the load balancer is getting stuck with a high error, but as it gradually eliminates possibilities it does tend to find a better result. This graph also shows that CFS was far less consistent than the methods proposed in this paper.

![Graphs showing standard deviation of error](image)

Figure 4.2: CPU time lost to load balancing

Figure 4.2 show the amount of processor time used by load balancing on all four cores. These graphs are in units of microseconds per second and represent the total time used on all cores.
Average Time Consumed: This graph shows the average time consumed by each load balancing method in units of microseconds per second. For the first five seconds method Sum, Max, and Bal use zero processing time. This is expected since; the load balancer is not enabled until second five of the test. After the time consumed spikes to about 1600 $\mu s/s$ as the load balancer tries to balance the initial state. Once it has found an arrangement the time consumes settle at about 500 $\mu s/s$, and again spikes when core 2 switches to full performance at second thirty. The three proposed methods performed similarly, but CFS uses much less time. The CFS load balancer has a cut point built in, so that if it does not quickly find a solution it simply stops running.

Maximum Time Consumed: This is the most time the load balancer consumed for each method at any given measurement. Again, this graph is the total for all four cores and is measured in microseconds of processing time per second of real time. The most any method used is about 20000 $\mu s/s$ which is 5000 $\mu s/s/core$ or 0.5% of the available processing time. The Bal method has the worst spikes, but overall the methods performance similarly. Again, the CFS load balancer performs the best.

Standard Deviation of Time Consumed: This graph shows the deviation amongst the data points. There is more deviation in the time consumed than there is in the error, but still reasonable.

4.4 Test 2

Test 2 demonstrates the performance of the load balancer on a system where tasks are frequently entering and leaving the system. In a real environment, tasks will perform some amount of input/output operations that will cause blocking. Like Test 1 this test is asymmetric. Processor cores 0 and 1 start at full speed and
remain at full speed for the duration of the test. Core 2 will start at half speed and switch to full speed thirty seconds into the test, and core 3 will start and remain at half speed. This test uses the same task as Test 1, which performs simple math operations in a loop. In this test however, the tasks sleep and run for random amount of time, but following the same schedule for each trail. Each task sleeps about 60% of time. Running for 1-400 ms and then sleeping for 1-600ms. The weights and counts of tasks used in this test are given in the following tables.

<table>
<thead>
<tr>
<th>Weight</th>
<th>100</th>
<th>200</th>
<th>250</th>
<th>300</th>
<th>450</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>1034</th>
<th>1096</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>23</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weight</th>
<th>2021</th>
<th>3000</th>
<th>3215</th>
<th>4637</th>
<th>5982</th>
<th>6045</th>
<th>6982</th>
<th>7901</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>5</td>
<td>7</td>
<td>12</td>
<td>20</td>
<td>18</td>
<td>8</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

Figure 4.3: Load balancer performance by error metric
**Average Error by Metric E and R:** On both metrics E and R, the three proposed methods perform similarly to each other. When the load balancer begins running, the error quickly drops from its starting state to a steady state that does not change very much. The static workload in Test 1 however, achieved an error much closer to zero. The CFS load balancer performs much worse than the proposed methods.

**Maximum Error by Metric E and R:** All method performed similarly by maximum error. The values of the maximum errors are likely to be dominated by the luck of which tasks are entering and leaving the system at any given time. The CFS load balancer should have an advantage here as it can choose which processors to place tasks on when they are reentering the system, while the proposed methods can only fixes the errors after the tasks have been placed. This is test though is asymmetric which the CFS load balancer is not designed for.

![Figure 4.4: CPU time lost to load balancing](image-url)
**Average Time Consumed:** The CFS load balancer consume the least time, but again it only check a couple of possibilities before giving up. All three of the proposed method perform consume between 1500 and 2000 $\mu s/s$ which is about 0.04% of the system processing time.

**Maximum Time Consumed:** For the maximum time consumed the proposed methods consumed about 10000 $\mu s/s$ or 0.25% of the available processing time. There is little difference between the proposed methods. CFS consumed the least amount of time.

### 4.5 Test 3

Test 1 and Test 2 were both asymmetric tests, but many systems are symmetric. For Test 3 all cores are set to run at there full performance. There are 147 tasks varying in weight from 101 to 7921. As they did in Test 2 the tasks will spend 60% of their time sleep. The tasks will run for 1-400 ms and sleep for 1-600 ms. Each task follows its own predetermined sleep/run schedule.

On the symmetric test, CFS slightly out performed the proposed methods. As on the previous tests, the proposed methods all performed similarly to each other. On this test however, the CFS load balancer performs inline with the proposed methods even achieves a better mean, worst case, and deviation by a small amount. The CFS load balancer uses a simpler method to balance the processors, which works well in the simpler symmetric case. The proposed methods are design to balance an asymmetric system, which has many scenarios that cannot happen on a symmetric system. This is shown in the design section, by the numerous cases in the proofs that can never be true on a symmetric system.
Figure 4.5: Load balancer performance by error metrics

Figure 4.6: CPU time lost to load balancing
As in the previous tests, the proposed methods all consume about the same amount of processing time. The CFS load balancer takes less time than the proposed methods.

4.6 Test 4

Test 4 is design to demonstrate a particular scenario. This test forces a start state where core 1 has 1 task with a weight of 4000, core 2 has a task of weight 3000 and a task with weight 2000, core 3 has 2 tasks with weight 2000, and core 4 has 30 tasks with weight 100. Thus, core 1 has total weight of 4000, core 2 has 5000, core 3 has 4000, and core 4 has 3000. This particular has multiple solutions with no error, where each processor has 4000 weight. The problem is that moving any one task will make the error worse than it is now, so greedy solutions that just move one task at a time will not find a solution.

Figure 4.7: Load balancer performance by error metrics
The CFS load balancer and the proposed methods find the solution. The proposed methods have rules that allow it to consider exchanging sets of tasks between two processors, which allow them to find the solution quickly. The CFS load balancer finds the solution, but it takes longer. This is because the CFS load balancer normally only considers the benefit of moving individual tasks, but occasionally moves an arbitrary task from the most loaded processors. In this example, moving any task from the over loaded processor would allow the greedy approach to find the solution. Thus, the CFS load balancer finds the solution, but takes longer.

Notice that the CFS load balancer does not get to zero on the Maximum error and deviation graphs. The CFS load balancer sometimes stops moving tasks with a few of the 100 weight tasks on the wrong processor. I do not know why the CFS load balancer sometimes leaves those tasks on the wrong processor.

![Figure 4.8: CPU time lost to load balancing](image-url)
On this test, none of the load balancing methods require much time. It is a simple scenario to solve, so long as the algorithm has a method to get past the greedy problem.
5 Conclusion

5.1 Results

This paper lays out methods for implementing a Proportional Share Scheduler in an asymmetric environment. By considering processors as producing performance and measuring the amount of performance assigned to a task instead of the amount of time, this paper provides a framework by which to measure the progress of PS tasks. This framework has the same results as the framework in [2] on symmetric systems, and additionally allows for fair allocation on asymmetric systems. Lag is tracked in performance units instead of time, so that results are independent of the performance of the processor. Furthermore tasks can be kept in balance by ensuring processors are assigned a total weight in proportion to their performance.

This paper also proposes a load balancer to ensure that each processor has the appropriate amount of weight. The error of the system and the tasks on the system is measured in terms of the error in the weight assignment of the processors. Two methods are proposed for measuring the error. The maximum error provides the maximum percent error on any processor, and is a measure of the worst-case scenario for any task in the system. The sum of the percent errors provides a single measurement to determine the error of the entire system. From the results of the test data, the two methods are very similar.

Three methods are proposed for a load balancer to ensure the processor have the correct amount of weight and to minimize the error of the system. From the test data, all three of the proposed methods provide effective load balancing for Asymmetric Proportional Share Scheduler systems. On each of the tests the three proposed methods performed nearly identically to each other, showing that there
is very little difference between. Performance was also similar regardless of which metric was chosen to measure the error showing that the two metrics are closely related. Since all three methods perform equally, the pair balancing method choice. Pair balancing balances two processors with respect to each other, instead of considering the error of the system as a whole. This means that pair balancing does not need to know the total weight of all the tasks in the system, and it does not need to know the total performance of the system. This provides for easy scalability, particularly in a clustering environment. Balancing occurring on one node does not require any knowledge of other nodes in the system. Balancing among nodes only requires nodes to have knowledge of the node that it is actively using to balance.

All three load balancing methods out performed the typical CFS load balancer in asymmetric scenarios. The reduced error from using any of these load balancing methods will help tasks receive shares of the processor closer to their ideal shares. By improving processor allocation, the predictability of these proportional share tasks will significantly improved.

5.2 Future Work

Additional rule sets should be considered. The balancer methods proposed in the paper all use the same set of rules for moving tasks between processors. The rules used in this paper do provide methods to exchange multiple tasks between nodes in one rule, which helps to keep the load balancer from being stuck, but other methods or rules should be considered. This could improve the ability of the load balancer to reduce the error of the system and reduce the amount of processing time. Additionally the load balancer should consider
stopping before the search is complete to reduce processing time when a reasonable solution has been found.

Cost should be associated with moves. Moving many tasks or moving tasks with large working sets can be expensive, particularly when moving among processors that use different memory pools. Predicting the cost of moving tasks would allow the load balancer to move tasks only when an appropriate amount of benefit will be gained from the move. This not only saves the load balancer processing time, but it also reduces the amount of time tasks spend retrieving their working sets.

Asymmetric proportional share load balancing could also be used to reduce power consumption. Existing power management systems reduce the performance of processors to save power. This can cause the system to become asymmetric, thus an asymmetric load balancer was necessary for proportional share scheduler to run such systems, but using such a load balancer could further benefit power management. Currently power management has to guess how busy a processor is by its idling time. Proportional share systems however, can be designed to know how much total load the system is intended to have. Thus a load balancer could determine how loaded system and reduce the performance and power consumption without ever idling the processor.
References


Appendix A: Proof of Criteria for Metric E

Proof. Proof of Theorem 3.3.7

\[ \forall X \subseteq Q(k), Y \subseteq Q(l) \text{ where } W(Y) > W(X) \geq 0. \]  
\[ X \text{ is the set of tasks being moved from processor } k \text{ to processor } l, \]  
\[ \text{and similarly } Y \text{ is the set of tasks being moved from processor } l \text{ to processor } k. \]  
\[ \text{If the load balancer is making a change then } Y \neq \emptyset. \]

\[ W(Y) > W(X) \implies W(Y) - W(X) > 0 \]
\[ r_k(Q(k) \setminus X \cup Y) = \frac{[W(Q(k)) + W(Y) - W(X)]}{p(k)S} - 1 \]
\[ > \frac{[W(Q(k))]P}{p(k)S} - 1 \]
\[ r_k(Q(k) \setminus X \cup Y) > r_k(Q(k)) \]
\[ r_l(Q(l) \setminus Y \cup X) = \frac{[W(Q(l)) - W(Y) + W(X)]}{p(k)S} - 1 \]
\[ < \frac{[W(Q(l))]P}{p(k)S} - 1 \]
\[ r_l(Q(l) \setminus Y \cup X) < r_l(Q(l)) \]

This proof is broken out into cases based on the sign of each of the following cases: \( r_k(Q(k)), r_l(Q(l)), r_k(Q(k) \setminus X \cup Y), \) and \( r_l(Q(l) \setminus Y \cup X). \) The absolute value of those functions is used to calculate the error. The cases allow the absolute values to be removed from the inequalities. Each case name starts with a number (1-4), this number is consistent with the signs of \( r_k(Q(k)) \) and \( r_l(Q(l)). \) Next the case name has a letter (a-d) which is consistent with the signs of \( r_k(Q(k) \setminus X \cup Y) \) and \( r_l(Q(l) \setminus Y \cup X). \) In some of the situations, more than one case can be shown with the same proof, in this situation the case name will have all of the letters for the cases it is demonstrating. Cases 2a and 2d are dependent on which processor has better performance these cases are split using an extra number: 2a1, 2a2, 2d1, 2d2.
A.1 Case 1

Case 1 has the assumption that \( r_k(Q(k)) > 0 \) and \( r_l(Q(l)) \leq 0 \). All subcases a-d are identical. This case considers the possibility of moving weight from an underloaded processor to an overloaded processor. As would be expected moving weight from \( l \) to \( k \) in this case will always increase the error metric \( E \). Thus, the load balancer should not do anything in this situation. To demonstrate this consider the effects moving weight, on the functions \( r_k \) and \( r_l \).

\[
|r_k(Q(k) \cup X \cup Y)| + |r_l(Q(l) \cup Y \cup X)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right| + \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|
\]

Since \( W(Y) - W(X) > 0 \)

\[
= \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 + \frac{[W(Y) - W(X) - W(Q(l))]P}{p(l)S} + 1
\]

Since \( W(Y) - W(X) > 0 \)

\[
> \frac{[W(Q(k))]P}{p(k)S} - 1 + \frac{[-W(Q(l))]P}{p(l)S} + 1
\]

\[
> r_k(Q(k)) + -r_l(Q(l))
\]

\[
|r_k(Q(k) \cup X \cup Y)| + |r_l(Q(l) \cup Y \cup X)| > |r_k(Q(k))| + |r_l(Q(l))|
\]

The error \( E \) is increased by moving weight from \( l \) to \( k \), thus the load balancer should not commit a move in this case.

A.2 Case 2

Case 2 assumes \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \). Each subcase a-d has it own subsection, some of which are further broken into two parts based on processor performance.
A.2.1 Case 2a

Subcase 2a assumes adds the additional assumption \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus Y \cup X) > 0 \). This subcase is additionally split into two more cases the first assumes \( p(l) \leq p(k) \) and the second assumes the opposite, \( p(l) > p(k) \).

A.2.1.1 Case 2a1

**Assumptions:** \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus Y \cup X) > 0 \) and \( p(l) \leq p(k) \).

In this case the pair of processors \( l \) and \( k \) have more weight than the two of them can run, which implies there must be at least one other processor in the system. Since \( k \) is the faster processor, it encounters a smaller relative overload per unit weight. The extra weight should be moved from \( l \), where it currently is, to \( k \), thereby reducing the total relative overload for \( k \) and \( l \).

\[
|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)|
= \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right| + \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|
= \frac{W(Q(k))P}{p(k)S} - 1 + \frac{W(Q(l))P}{p(l)S} - 1
+ \frac{[W(Y) - W(X)]P}{p(l)S} + \frac{[W(Q(k)) - W(Y) - W(X)]P}{p(k)S}
= \frac{W(Y) - W(X)}{p(k)p(l)} - 1 + \frac{W(Q(k))P}{p(k)S} - 1 + \frac{W(Q(l))P}{p(l)S} - 1
= \frac{[W(Y) - W(X)](p(l) - p(k))P}{p(k)p(l)S} + \frac{W(Q(k))P}{p(k)S} - 1 + |r_i(Q(l))|\]
Since $p(l) - p(k) \leq 0$ and $W(Y) - W(X) > 0$ the first term is negative or zero.

$$|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| \leq 0 + \frac{W(Q(k))p}{p(k)S} - 1 + |r_l(Q(l))|$$

The error is reduced by moving weight to $k$. Note that the error will remain unchanged if $p(k) = p(l)$ and $r_k(Q(k)) = 0$, moreover if simply $p(k) = p(l)$ it makes no difference which processor bares the overload.

**A.2.1.2 Case 2a2**

**Assumptions:** $r_k(Q(k)) \leq 0$ and $r_l(Q(l)) > 0$ and $r_k(Q(k) \setminus X \cup Y) > 0$ and $r_l(Q(l) \setminus Y \cup X) > 0$ and $p(l) > p(k)$.

Just like case 2a1 there is too much weight for the pair of processor, thus the system must have at least one other processor. The difference is in this case, processor $l$ is faster than processor $k$, so while weight should be moved to $k$ to fix its under load the extra weight should remain on $l$.

The error will decrease iff $W(Y) - W(X) < 2\left(\frac{\alpha_p(l)S - p(l)W(Q(k))}{p(l) - p(k)}\right)$. Recall that $\alpha_n$ is $\frac{p(N)}{p}$. This case starts by manipulating the error function for the end state into a more
useful form.

\[
|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| \\
= \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right| + \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right| \\
= \frac{W(Q(k))P}{p(k)S} + \frac{[W(Y) - W(X)]P}{p(k)S} + \frac{W(Q(l))P}{p(l)S} - \frac{[W(Y) - W(X)]P}{p(l)S} - 2 \\
= \frac{[W(Y) - W(X)]p(l)P}{p(k)p(l)S} + \frac{W(Q(k))p(k)P}{p(k)S} - 1 + |r_l(Q(l))| \\
\]

Now the criteria is demonstrated by first showing that

\[
W(Y) - W(X) < 2 \left( \frac{\alpha \cdot p(l)S - p(l)W(Q(k))}{p(l) - p(k)} \right) \text{ reduce the error } E \text{ and then showing that} \\
W(Y) - W(X) \geq 2 \left( \frac{\alpha \cdot p(l)S - p(l)W(Q(k))}{p(l) - p(k)} \right) \text{ does not decrease the error } E.
\]

**Part 1:** \( W(Y) - W(X) < 2 \left( \frac{\alpha \cdot p(l)S - p(l)W(Q(k))}{p(l) - p(k)} \right) \)

\[
|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| \\
< 2 \left( \frac{\alpha \cdot p(l)S - p(l)W(Q(k))}{p(l) - p(k)} \right) \left( p(l) - p(k) \right) \frac{P}{S} + \frac{W(Q(k))p(k)P}{p(k)S} - 1 + |r_l(Q(l))| \\
< \frac{2p(k)p(l)}{p(k)p(l)S} - \frac{2p(l)W(Q(k))p(k)S}{p(k)p(l)S} + \frac{W(Q(k))p(k)P}{p(k)S} - 1 + |r_l(Q(l))| \\
< 2 - \frac{W(Q(k))P}{p(k)S} + \frac{W(Q(k))p(k)P}{p(k)S} - 1 + |r_l(Q(l))| \\
< - \frac{W(Q(k))P}{p(k)S} + 1 + |r_l(Q(l))| \\
< - r_k(Q(k)) + |r_l(Q(l))| \\
\]

\[
|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| < |r_k(Q(k))| + |r_l(Q(l))| \\
\]
Part 2: $W(Y) - W(X) \geq 2\left(\frac{a_{l}p(l)S - p(l)W(Q(k))}{p(l) - p(k)}\right)$

\[
|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| \\
\geq \frac{2\left(\frac{a_{l}p(l)S - p(l)W(Q(k))}{p(l) - p(k)}\right)(p(l) - p(k))P}{p(k)p(l)S} + \frac{W(Q(k))P}{p(k)S} - 1 + |r_l(Q(l))| \\
\geq \frac{2p(k)p(l)S}{p(k)p(l)S} - \frac{2p(l)W(Q(k))P}{p(k)p(l)S} + \frac{W(Q(k))P}{p(k)S} - 1 + |r_l(Q(l))| \\
\geq 2 - 2\frac{W(Q(k))P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1 + |r_l(Q(l))| \\
\geq - \frac{W(Q(k))P}{p(k)S} + 1 + |r_l(Q(l))| \\
\geq - r_k(Q(k)) + |r_l(Q(l))| \\
\]

\[
|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| \geq |r_k(Q(k))| + |r_l(Q(l))| \\
\]

The error will be improved iff $W(Y) - W(X) < 2\left(\frac{a_{l}p(l)S - p(l)W(Q(k))}{p(l) - p(k)}\right)$.

A.2.2 Case 2b

Assumptions: $r_k(Q(k)) \leq 0$ and $r_l(Q(l)) > 0$ and $r_k(Q(k) \setminus X \cup Y) \leq 0$ and $r_l(Q(l) \setminus Y \cup X) > 0$.

In this case the processor $k$ is under loaded processor $l$ and the load balancer is considering moving an amount of weight small enough that $k$ will remain under loaded and $l$ will remain overloaded. This will always reduce the error. To demonstrate this consider the effects of moving a small amount of weight.
when $k$ is under loaded and $l$ is overloaded.

$$
|r_k(\{Q(k)\} \cup Y)| + |r_l(\{Q(l)\} \cup X)| = \left| \frac{W(\{Q(k)\}) + W(Y) - W(X)}{p(k)S} - 1 \right| + \left| \frac{W(\{Q(l)\}) - (W(Y) - W(X))}{p(l)S} - 1 \right|
$$

$$
=-\frac{\left|W(Y) - W(X)\right|}{p(k)S} - \frac{\left|W(Y) - W(X)\right|}{p(l)S} - \frac{W(Q(k))}{p(k)S} + 1 + \frac{W(Q(l))}{p(l)S} - 1
$$

$$
=\frac{-[W(Y) - W(X)]p(l)S}{p(k)p(l)S} - \frac{W(Y) - W(X)p(k)P}{p(k)p(l)S} - r_k(Q(k)) + r_l(Q(l)) + |r_k(Q(k))| + |r_l(Q(l))|
$$

Since the first term is strictly negative,

$$
|r_k(\{Q(k)\} \cup Y)| + |r_l(\{Q(l)\} \cup X)| < |r_k(Q(k))| + |r_l(Q(l))|
$$

The error will always improve by moving weight from processor $l$ to processor $k$ when $l$ remains overloaded and $k$ remains under loaded.

### A.2.3 Case 2c

**Assumptions:** $r_k(Q(k)) \leq 0$ and $r_l(Q(l)) > 0$ and $r_k(Q(k) \cup Y > 0$ and $r_l(Q(l) \cup X) \leq 0$.

In this case, the load balancer is considering reversing the situation. That is processor $k$ starts under load and ends overloaded while processor $l$ starts overload and ends under loaded. More than the ideal amount of weight is being moved, so this case considers how much weight can be moved before the error is worse than it was at the start.

The error will be reduce by making the move iff $W(Y) - W(X) < 2\frac{p(k)W(Q(k)) - p(l)W(Q(k))}{p(k)+p(l)}$

To demonstrate this, the error function of the end state is first manipulated into a more useful form. Then the criteria are broken into its two parts. $E$ is reduced if $W(Y) - W(X) < 2\frac{p(k)W(Q(k)) - p(l)W(Q(k))}{p(k)+p(l)}$ and $E$ will not decrease if
\[ W(Y) - W(X) \geq 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}. \] Each part is shown separately.

\[ |r_k(Q(k) \setminus X \cup Y) + |r_l(Q(l) \setminus Y \cup X)| \]
\[ = \left| \frac{W(Q(k)) + W(Y) - W(X)}{p(k)S} - 1 \right| + \left| \frac{W(Q(l)) - W(Y) + W(X)}{p(l)S} - 1 \right| \]
\[ = \left| \frac{W(Y) - W(X)}{p(k)S} + \frac{W(Y) - W(X)}{p(l)S} + \frac{W(Q(k))}{p(k)S} - \frac{W(Q(l))}{p(l)S} \right| + 1 \]
\[ = \frac{|W(Y) - W(X)|}{p(k)p(l)S} + \frac{W(Q(k))}{p(k)S} - \frac{W(Q(l))}{p(l)S} \]

**Part 3:** \( W(Y) - W(X) < 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)} \)

\[ < \left( \frac{p(k) + p(l)}{p(k)p(l)S} \right) \left[ p(k) + p(l) \right] \frac{W(Q(k))}{p(k)S} - \frac{W(Q(l))}{p(l)S} \]
\[ < \left( \frac{2[p(k)W(Q(l)) - p(l)W(Q(k))]}{p(k)p(l)S} \right) + \frac{W(Q(k))}{p(k)S} - \frac{W(Q(l))}{p(l)S} \]
\[ < \frac{2p(k)W(Q(l)) - 2p(l)W(Q(k)) + p(l)W(Q(k)) - p(k)W(Q(l))}{p(k)p(l)S} \]
\[ < \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k)p(l)S} \]
\[ < \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k)p(l)S} \]
\[ < \frac{W(Q(l))}{p(l)S} - 1 - \frac{W(Q(k))}{p(k)S} + 1 \]
\[ < |r_l(Q(l)) - r_k(Q(k))| \]
\[ < |r_l(Q(l))| + |r_k(Q(k))| \]

\[ |r_k(Q(k) \setminus X \cup Y) + |r_l(Q(l) \setminus Y \cup X)| < |r_l(Q(l))| + |r_k(Q(k))|. \]

The end error is reduced if \( W(Y) - W(X) < 2 \frac{p(k)W(Q(l)) - p(l)W(Q(k))}{p(k) + p(l)}. \)
Part 4: \( W(Y) - W(X) \geq 2^{p(k)W(Q(l)) - p(l)W(Q(k)) \over p(k) + p(l)} \)

\[
\begin{align*}
&\geq 2^{p(k)W(Q(l)) - p(l)W(Q(k)) \over p(k) + p(l)} [p(k) + p(l)]P + W(Q(k))P \over p(k)S - W(Q(l))P \over p(l)S \\
&\geq 2[p(k)W(Q(l)) - p(l)W(Q(k))]P \over p(k)p(l)S + p(l)W(Q(k))P \over p(k)p(l)S - p(k)W(Q(l))P \over p(k)p(l)S \\
&\geq 2p(k)W(Q(l))P - 2p(l)W(Q(k))P + p(l)W(Q(k))P - p(k)W(Q(l))P \over p(k)p(l)S \\
&\geq p(k)W(Q(l))P - p(l)W(Q(k))P \over p(k)p(l)S \\
&\geq p(k)W(Q(l))P - p(l)W(Q(k))P \over p(k)p(l)S \\
&\geq W(Q(l))P \over p(l)S - 1 - W(Q(k))P \over p(k)S + 1 \\
&\geq r_i(Q(l)) - r_k(Q(k)) \\
&\geq |r_i(Q(l))| + |r_k(Q(k))|
\end{align*}
\]

\(|r_k(Q(k) \setminus X \cup Y)| + |r_i(Q(l) \setminus Y \cup X)| \geq |r_i(Q(l))| + |r_k(Q(k))|.

The error will not be reduced if \( W(Y) - W(X) \geq 2^{p(k)W(Q(l)) - p(l)W(Q(k)) \over p(k) + p(l)} \).

A.2.4 Case 2d

A.2.4.1 Case 2d1

Assumptions: \( r_k(Q(k)) \leq 0 \) and \( r_i(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) \leq 0 \) and \( r_i(Q(l) \setminus Y \cup X) \leq 0 \) and \( p(k) \leq p(l) \)

In this case the pair of processors \( k \) and \( l \) do not have their fair share of the weight and as such some other processors must exist in the system. Since processor \( l \) is faster it will have less relative error for the same amount of weight shortage. Thus the error will be minimized by getting processor \( k \) as close to its ideal weight as possible. The result is that the error will always be reduced in this case.
error there is a limit to the amount of weight that should be moved away from
processor must exist in the system. In this case however, processor k

\[ A.2.4.2 \text{ Case 2d2} \]

\[ \left| r_{k}(Q(k) \setminus X \cup Y) + r_{l}(Q(l) \setminus Y \cup X) \right| = \left| \frac{W(Q(k)) + W(Y) - W(X)}{p(k)S} - 1 \right| + \left| \frac{W(Q(l)) - W(Y) + W(X)}{p(l)S} - 1 \right| \]

\[ = \left| \frac{-(W(Y) - W(X)) - W(Q(k))}{p(k)S} \right| + 1 + \left| \frac{W(Y) - W(X) - W(Q(l))}{p(l)S} \right| + 1 \]

\[ = \frac{-(W(Y) - W(X))p(l)S}{p(k)p(l)S} + \frac{W(Y) - W(X)p(k)S}{p(k)p(l)S} - r_{k}(Q(k)) - r_{l}(Q(l)) \]

\[ = \frac{W(Y) - W(X)[p(k) - p(l)]S}{p(k)p(l)S} + |r_{k}(Q(k))| - r_{l}(Q(l)) \]

Since \( p(k) - p(l) \leq 0 \) and \( W(Y) - W(X) > 0 \) the first term is negative or zero.

\[ |r_{k}(Q(k) \setminus X \cup Y)| + |r_{l}(Q(l) \setminus Y \cup X)| \leq 0 + |r_{k}(Q(k))| - r_{l}(Q(l)) \]

Since \( -r_{l}(Q(l)) \leq |r_{l}(Q(l))| \)

\[ |r_{k}(Q(k) \setminus X \cup Y)| + |r_{l}(Q(l) \setminus Y \cup X)| \leq |r_{k}(Q(k))| + |r_{l}(Q(l))| \]

It should be noted that if \( p(k) = p(l) \) it does not matter which processor bares the
shortage, the error will be the same.

A.2.4.2 Case 2d2

**Assumptions:** \( r_{k}(Q(k)) \leq 0 \) and \( r_{l}(Q(l)) > 0 \) and \( r_{k}(Q(k) \setminus X \cup Y) \leq 0 \) and
\( r_{l}(Q(l) \setminus Y \cup X) \leq 0 \) and \( p(k) > p(l) \).

Just like case 2d1 the pair does not have enough weight, thus some other
processor must exist in the system. In this case however, processor k is faster, so
there is a limit to the amount of weight that should be moved away from l. The
error \( E \) will be improved iff \( W(Y) - W(X) < 2 \frac{p(k)W(Q(l) - p(k))S}{p(k) - p(l)} \). The error function
of the end state is first manipulated into a more friendly form. Then the positive part
of the criteria is demonstrated, \( E \) is reduced if \( W(Y) - W(X) < 2 \frac{p(k)W(Q(l) - p(k))S}{p(k) - p(l)} \).
Followed by the negative part, $E$ is not reduced if $W(Y) - W(X) \geq 2\frac{p(k)W(Q(l)) - p(k)|S|}{p(k) - p(l)}$.

\[
|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)|
= \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right| + \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1
\]
\[
= \left| \frac{-(W(Y) - W(X)) - W(Q(k))]P}{p(k)S} + 1 \right| + \frac{[W(Y) - W(X)) - W(Q(l))]P}{p(l)S} + 1
\]
\[
= \frac{[W(Y) - W(X))p(l)P}{p(k)p(l)S} + \frac{[W(Y) - W(X)]p(k)P}{p(k)p(l)S} - r_k(Q(k)) - \frac{W(Q(l))]P}{p(l)S} + 1
\]
\[
= \frac{[W(Y) - W(X)][p(k) - p(l)]P}{p(k)p(l)S} + |r_k(Q(k))| - \frac{W(Q(l))]P}{p(l)S} + 1
\]

\textbf{Part 5:} $W(Y) - W(X) < 2\frac{p(k)W(Q(l)) - p(k)|S|}{p(k) - p(l)}$

\[
< \frac{2p(k)W(Q(l)) - p(k)|S|}{p(k) - p(l)} + |r_k(Q(k))| - \frac{W(Q(l))]P}{p(l)S} + 1
\]
\[
< \frac{2p(k)PW(Q(l)) - 2p(k)|S|}{p(k)p(l)S} + 1 + |r_k(Q(k))|
\]
\[
< \frac{p(k)PW(Q(l))}{p(k)p(l)S} - 2 + 1 + |r_k(Q(k))|
\]
\[
< |r_l(Q(l)) + |r_k(Q(k))|
\]
\[
< |r_l(Q(l))| + |r_k(Q(k))|
\]

$|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| < |r_l(Q(l))| + |r_k(Q(k))|

The error, $E$, decreases if $W(Y) - W(X) < 2\frac{p(k)W(Q(l)) - p(k)|S|}{p(k) - p(l)}$. 

\textbf{Part 6:} \( W(Y) - W(X) \geq 2 \frac{p(k) W(Q(l)) - p(k) \alpha l_S}{p(k) - p(l)} \)

\[
\geq \frac{2p(k) W(Q(l)) - 2p(k)p(l) S}{p(k)p(l) S} - \frac{p(k) P W(Q(l))}{p(k)p(l) S} + 1 + |r_k(Q(k))|
\]

\[
\geq |r_l(Q(l))| + |r_k(Q(k))|
\]

\[
\geq |r_l(Q(l))| + |r_k(Q(k))|
\]

\[
|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| \geq |r_l(Q(l))| + |r_k(Q(k))|
\]

The error does not decrease if \( W(Y) - W(X) \geq 2 \frac{p(k) W(Q(l)) - p(k) \alpha l_S}{p(k) - p(l)} \).

\textbf{A.3 Case 3}

\textbf{A.3.1 Case 3a}

\textbf{Assumptions:} \( r_k(Q(k)) > 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus Y \cup X) > 0 \).

Both processors start overloaded and both processors are overloaded at the end. This is very similar to case 2a1 where both processors end overloaded. The end result is the same; to minimize the error, \( E \), all of the overload should be moved to the faster processor.

The error will be decreases by moving weight to \( k \) iff \( p(k) > p(l) \).

To show this the end state error function is manipulated into a form where the affects of the processor speed are more apparent. Then it is demonstrated that the
error $E$ will reduce if $p(k) > p(l)$ and that $E$ will not reduce if $p(k) \leq p(l)$.

$$|r_k(Q(k) \setminus X \cup Y) + r_l(Q(l) \setminus Y \cup X)|$$

$$= \left| \frac{[W(Q(k)) + W(Y) - W(X)]}{p(k)S} - 1 \right| + \left| \frac{[W(Q(l)) - W(Y) + W(X)]}{p(l)S} - 1 \right|$$

$$= \frac{W(Q(k))}{p(k)S} + \frac{W(Y) - W(X)}{p(k)S} + \frac{W(Q(l))}{p(l)S} - \frac{W(Y) - W(X)}{p(l)S} - 2$$

$$= \frac{W(Y) - W(X)l}{p(k)p(l)S} - \frac{W(Y) - W(X)p(l)}{p(k)p(l)S} + \frac{W(Q(k))p(l)}{p(k)S} + \frac{W(Q(l))p(l)}{p(l)S} - 2$$

$$= \frac{W(Y) - W(X)l}{p(k)p(l)S} + |r_k(Q(k))| + |r_l(Q(l))|$$

**Part 7:** $p(k) > p(l)$

$p(l) - p(k) < 0$ therefore the first term is strictly negative.

$$|r_k(Q(k) \setminus X \cup Y) + r_l(Q(l) \setminus Y \cup X)| <$$

$$0 + |r_k(Q(k))| + |r_l(Q(l))||r_k(Q(k) \setminus X \cup Y) + |r_l(Q(l) \setminus Y \cup X)| < |r_k(Q(k))| + |r_l(Q(l))|$$

$E$ always decreases if $p(k) > p(l)$.

**Part 8:** $p(k) \leq p(l)$

$p(l) - p(k) \leq 0$ therefore the first term is never negative.

$$|r_k(Q(k) \setminus X \cup Y) + r_l(Q(l) \setminus Y \cup X)| \geq$$

$$0 + |r_k(Q(k))| + |r_l(Q(l))||r_k(Q(k) \setminus X \cup Y) + |r_l(Q(l) \setminus Y \cup X)| \geq |r_k(Q(k))| + |r_l(Q(l))|$$

$E$ never decreases if $p(k) \leq p(l)$.

**A.3.2 Case 3b,d**

**Assumptions:** $r_k(Q(k)) > 0$ and $r_l(Q(l)) > 0$ and $r_k(Q(k) \setminus X \cup Y) \leq 0$.

The case is impossible as it would imply that an increase in the total weight on processor $k$ would actually decrease its load. The following demonstrates that
\[ r_k(Q(k) \setminus X \cup Y) > 0 \text{ must be true.} \]

\[
r_k(Q(k) \setminus X \cup Y) = \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1
= \frac{W(Q(k))P^*}{p(k)S} - 1 + [w(Y) - W(X)]\frac{P}{p(k)}S
= r_k(Q(k)) + [W(Y) - W(X)]\frac{P}{p(k)}S
\]

Since \( r_k(Q(k)) > 0 \)
\[ > 0 + [W(Y) - W(X)]\frac{P}{p(k)}S \]

Since \( W(Y) - W(X) > 0 \)
\[ > 0 + [0]\frac{P}{p(k)}S \]

\[ r_k(Q(k) \setminus X \cup Y) > 0 \]

Thus, this case cannot happen.

A.3.3 Case 3c

**Assumptions:** \( r_k(Q(k)) > 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and
\( r_l(Q(l) \setminus Y \cup X) \leq 0 \).

Since both processors start overloaded this case implies that the system has
more than two processors. In the end state of this case processor \( l \) is under loaded,
thus this case must limit how much \( l \) is under loaded.
The error, \( E \), decreases iff \( W(Y) - W(X) < \frac{2p(k)W(Q(l))}{p(k)+p(l)} - 2p(k)\alpha l \).
This is demonstrated by first manipulating the end state error so that the term
\( W(Y) - W(X) \) can easily be substituted. Then breaking the criteria into a positive
part \( E \) is reduced if \( W(Y) - W(X) < \frac{2p(k)W(Q(l))}{p(k)+p(l)} - 2p(k)\alpha l \), and its negative part \( E \) is not
reduced if \( W(Y) - W(X) < \frac{2p(l)W(Q(l)) - 2p(k)\alpha_1 S}{p(k) + p(l)} \).

\[
|r_k(Q\setminus X \cup Y)| + |r_l(Q\setminus Y \cup X)| \\
= \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right| + \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right| \\
= \frac{[W(Y) - W(X)]P}{p(k)S} + \frac{[W(Y) - W(X)]P}{p(l)S} + \frac{W(Q(k))P}{p(k)S} - 1 - \frac{W(Q(l))P}{p(l)S} + 1 \\
= \frac{[W(Y) - W(X)]P[p(k) + p(l)]}{p(k)p(l)S} + \frac{W(Q(k))P}{p(l)S} - 1 - \frac{W(Q(l))P}{p(l)S} + 1 \\
= \frac{[W(Y) - W(X)]P[p(k) + p(l)]}{p(k)p(l)S} + |r_k(Q(k))| - \frac{W(Q(l))P}{p(l)S} + 1
\]

**Part 9:** \( W(Y) - W(X) < \frac{2p(l)W(Q(l)) - 2p(k)\alpha_1 S}{p(k) + p(l)} \)

\[
< \frac{2p(k)W(Q(l)) - 2p(k)\alpha_1 SP}{p(k)p(l)S} + |r_k(Q(k))| - \frac{W(Q(l))P}{p(l)S} + 1 \\
< \frac{2p(k)W(Q(l))P}{p(k)p(l)S} - 2p(k)PW(Q(l)) + |r_k(Q(k))| + 1 \\
< \frac{p(k)PW(Q(l))}{p(k)p(l)S} - 1 + |r_k(Q(k))| \\
< \frac{W(Q(l))P}{p(l)S} - 1 + |r_k(Q(k))| \\
< r_l(Q(l)) + |r_k(Q(k))|
\]

\[
|r_k(Q\setminus X \cup Y)| + |r_l(Q\setminus Y \cup X)| < |r_l(Q(l))| + |r_k(Q(k))|
\]

\( E \) is reduced if \( W(Y) - W(X) < \frac{2p(l)W(Q(l)) - 2p(k)\alpha_1 S}{p(k) + p(l)} \).
Part 10: \( W(Y) - W(X) \geq \frac{2p(k)W(Q(l)) - 2p(k)\alpha_{i}S}{p(k) + p(l)} \)

\[
\geq \frac{2p(k)W(Q(l)) - 2p(k)\alpha_{i}SP}{p(k)p(l)S} + |r_{k}(Q(k))| - \frac{W(Q(l))P}{p(l)S} + 1
\]

\[
\geq \frac{2p(k)W(Q(l))P - 2p(k)\alpha_{i}SP}{p(k)p(l)S} + |r_{k}(Q(k))| - \frac{W(Q(l))P}{p(l)S} + 1
\]

\[
\geq \frac{p(k)PW(Q(l))}{p(k)p(l)S} - 1 + |r_{k}(Q(k))|
\]

\[
\geq \frac{W(Q(l))P}{p(l)S} - 1 + |r_{k}(Q(k))|
\]

\[
\geq r_{i}(Q(l)) + |r_{k}(Q(k))|
\]

\[|r_{k}(Q(k) \setminus X \cup Y)| + |r_{i}(Q(l) \setminus Y \cup X)| \geq |r_{i}(Q(l))| + |r_{k}(Q(k))|\]

\( E \) is not reduced if \( W(Y) - W(X) \geq \frac{2p(k)W(Q(l)) - 2p(k)\alpha_{i}S}{p(k) + p(l)} \).

A.4 Case 4

A.4.1 Case 4a,b

Assumptions: \( r_{k}(Q(k)) \leq 0 \) and \( r_{i}(Q(l)) \leq 0 \) and \( r_{i}(Q(l) \setminus Y \cup X) > 0 \).

This is another impossible case, as it would imply that remove weight from processor was increasing its load. The following demonstrates that \( r_{i}(l \setminus Y \cup U) < 0 \) must be true.

\[
r_{i}(Q(l) \setminus Y \cup X) = \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1
\]

\[
= \frac{W(Q(l))P}{p(l)S} - 1 - \frac{W(Y) - W(X)}{p(l)S}
\]

\[
= r_{i}(Q(l)) - \frac{W(Y) - W(X)}{p(l)S}
\]

Since \( r_{i}(Q(l)) < 0 \)

\[
r_{i}(Q(l) \setminus Y \cup X) < 0 - \frac{W(Y) - W(X)}{p(l)S}
\]
Since \( W(Y) - W(X) > 0 \)
\[ r_l(Q(l) \setminus Y \cup X) < 0 - 0 \]
\[ r_l(Q(l) \setminus Y \cup X) < 0 \] Therefore, this case cannot happen.

### A.4.2 Case 4c

**Assumptions:** \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) \leq 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and
\[ r_l(Q(l) \setminus Y \cup X) \leq 0. \]

This case starts with both processors being underloaded; therefore, there must be more processors in the system. In addition, since there is not enough weight for both processors, the error will be minimized if the faster processor bares the shortage. In this case the error will decrease iff \( W(Y) - W(X) < \frac{2n_{p(l)}s - 2p(l)w(Q(k))}{p(k) + p(l)} \). To show this the end state error function is manipulated into a form where it is easy to substitute the term \( W(Y) - W(X) \). Then the criteria is broken into two parts \( E \) decreases if \( W(Y) - W(X) < \frac{2n_{p(l)}s - 2p(l)w(Q(k))}{p(k) + p(l)} \) and \( E \) does not decrease if \( W(Y) - W(X) \geq \frac{2n_{p(l)}s - 2p(l)w(Q(k))}{p(k) + p(l)} \). Both parts are shown separately.

\[
| r_k(Q(k) \setminus X \cup Y) | + | r_l(Q(l) \setminus Y \cup X) |
\]
\[
= \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right| + \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|
\]
\[
= \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 + \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} + 1
\]
\[
= \frac{[W(Y) - W(X)]P}{p(k)S} + \frac{[W(Y) - W(X)]P}{p(l)S} + \frac{W(Q(k))P}{p(k)S} - 1 - \frac{W(Q(l))P}{p(l)S} + 1
\]
\[
= \frac{[W(Y) - W(X)]p(l)P}{p(k)p(l)S} + \frac{[W(Y) - W(X)]p(k)P}{p(k)p(l)S} + \frac{W(Q(k))P}{p(k)S} - 1 - r_l(Q(l))
\]
\[
= \frac{[W(Y) - W(X)][p(k) + p(l)]P}{p(k)p(l)S} + \frac{W(Q(k))P}{p(k)S} - 1 + | r_l(Q(l)) |
\]
Part 11: $W(Y) - W(X) < \frac{2\alpha_k p(l)S - 2p(l)W(Q(k))}{p(k)p(l)}$ \\

\[
\begin{align*}
&\frac{2\alpha_k p(l)S - 2p(l)W(Q(k))}{p(k)p(l)}[p(k) + p(l)]P + \frac{W(Q(k))P}{p(k)S} - 1 + |r_l(Q(l))| \\
&\leq \frac{2\alpha_k p(l)S - 2p(l)W(Q(k))}{p(k)p(l)} + \frac{W(Q(k))P}{p(k)S} - 1 + |r_l(Q(l))| \\
&\leq 2 + \frac{-2p(l)W(Q(k))P}{p(k)p(l)S} + \frac{p(l)W(Q(k))P}{p(k)p(l)S} - 1 + |r_l(Q(l))| \\
&\leq -p(l)W(Q(k))P + 1 + |r_l(Q(l))| \\
&\leq - r_k(Q(k)) + |r_l(Q(l))|
\end{align*}
\]

$|r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| < |r_k(Q(k))| + |r_l(Q(l))|$

The error decreases if $W(Y) - W(X) < \frac{2\alpha_k p(l)S - 2p(l)W(Q(k))}{p(k)p(l)}$.

Part 12: $W(Y) - W(X) \geq \frac{2\alpha_k p(l)S - 2p(l)W(Q(k))}{p(k)p(l)}$ \\

\[
\begin{align*}
&\geq \frac{2\alpha_k p(l)S - 2p(l)W(Q(k))}{p(k)p(l)}[p(k) + p(l)]P + \frac{W(Q(k))P}{p(k)S} - 1 + |r_l(Q(l))| \\
&\geq \frac{2\alpha_k p(l)S - 2p(l)W(Q(k))}{p(k)p(l)} + \frac{W(Q(k))P}{p(k)S} - 1 + |r_l(Q(l))| \\
&\geq 2 + \frac{-2p(l)W(Q(k))P}{p(k)p(l)S} + \frac{p(l)W(Q(k))P}{p(k)p(l)S} - 1 + |r_l(Q(l))| \\
&\geq -p(l)W(Q(k))P + 1 + |r_l(Q(l))| \\
&\geq - r_k(Q(k)) + |r_l(Q(l))|
\end{align*}
\]
Each part is then shown using the sign of $p$

The error does not decrease if $W(Y) - W(X) \geq \frac{2\alpha_p l S - 2p(l)W(Q(k))}{p(k) + p(l)}$.

A.4.3 Case 4d

**Assumptions:** $r_k(Q(k)) \leq 0$ and $r_l(Q(l)) \leq 0$ and $r_k(Q(k) \setminus Y \cup X) \leq 0$ and $r_l(Q(l) \setminus Y \cup X) \leq 0$

In this case, both processors start under loaded, and both processors remain under loaded. Since both processors are under loaded, there must be another processor in the system. The error will be minimize by moving as much weight as possible to the slower processor, without overloading it.

The error, $E$, will decrease iff $p(l) > p(k)$.

To demonstrate this the end state error function is manipulated into so that the sign of $p(k) - p(l)$ can be easily examined. Then the criteria is broken into two parts the error reduces if $p(l) > p(k)$ and the error does not reduce if $p(l) \leq p(k)$.

Each part is then shown using the sign of $p(k) - p(l)$.

$$|r_k(Q(k) \setminus Y \cup X)| + |r_l(Q(l) \setminus Y \cup X)|$$

$$\frac{([W(Q(k)) + W(Y) - W(X)]p}{p(k)S} - 1) + \left| \frac{[W(Q(l)) - W(Y) + W(X)]p}{p(l)S} - 1 \right|$$

$$\frac{[-(W(Y) - W(X)) - W(Q(k))]p}{p(k)S} + 1 + \frac{[(W(Y) - W(X)) - W(Q(l))]p}{p(l)S} + 1$$

$$\frac{-[W(Y) - W(X)]p}{p(k)p(l)S} + \frac{[W(Y) - W(X)]p}{p(k)p(l)S} - r_k(Q(k)) - r_l(Q(l))$$

$$\frac{[W(Y) - W(X)][p(k) - p(l)]p}{p(k)p(l)S} + |r_k(Q(k))| + |r_l(Q(l))|$$

**Part 13:** $p(l) > p(k)$

Since $p(k) - p(l) < 0$ and $W(Y) - W(X) > 0$ the first term is strictly negative.

$$|r_k(Q(k) \setminus Y \cup X)| + |r_l(Q(l) \setminus Y \cup X)| <$$
Thus the error decreases if \( p(l) > p(k) \).

**Part 14: \( p(l) \leq p(k) \)**

Since \( p(k) - p(l) > 0 \) and \( W(Y) - W(X) > 0 \), the first term is never negative.

\[
0 + |r_k(Q(k))| + |r_l(Q(l))||r_k(Q(k) \setminus X \cup Y)| + |r_l(Q(l) \setminus Y \cup X)| \geq 0 + |r_k(Q(k))| + |r_l(Q(l))|
\]

Thus the error increases or remains unchanged if \( p(l) \leq p(k) \).
APPENDIX B: PROOF OF CRITERIA FOR METRIC R

Proof. of Theorem 3.3.21

B.1 Case 1

Assumptions: \( r_k(Q(k)) > 0 \) and \( r_l(Q(l)) \leq 0 \)

In this case, the maximum error is always increased by performing the move.

To show this examine the error of both \( r_k(Q(k) \setminus X \cup Y) \) and \( r_l(Q(l) \setminus Y \cup X) \) and realizing that they both increase.

\[
 r_k(Q(k) \setminus X \cup Y) = \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \\
= \frac{[W(Y) - W(X)]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1 \\
\text{Since } W(Y) - W(X) > 0 \\
> 0 + r_k(Q(k)) \\

r_k(Q(k) \setminus X \cup Y) > r_k(Q(k)) > 0 \\
|r_k(Q(k) \setminus X \cup Y)| > |r_k(Q(k))|
\]

\[
 r_l(Q(l) \setminus Y \cup X) = \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \\
= - \frac{[W(Y) - W(X)]P}{p(l)S} + \frac{W(Q(l))P}{p(l)S} - 1 \\
\text{Since } W(Y) - W(X) > 0 \\
< - 0 + r_l(Q(l)) \\

r_l(Q(l) \setminus Y \cup X) < r_l(Q(l)) \leq 0 \\
|r_l(Q(l) \setminus Y \cup X)| > |r_l(Q(l))|
\]

\[ \therefore \max(|r_k(Q(k) \setminus X \cup Y)|, |r_l(Q(l) \setminus Y \cup X)|) > \max(|r_k(Q(k)|, |r_l(Q(l)|) \]

The error is always increased.
B.2 Case 2

B.2.1 Case 2a

Assumptions: \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus X \cup Y) > 0 \).

In this case the max error of the pair will be reduced iff \( W(Y) - W(X) < \max\left( W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), 2[\alpha_k S - W(Q(k))] \right) \).

By Lemma 3.3.19, \( |r_l(Q(l) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|) \).

To show that \( |r_k(Q(k) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|) \) consider \( r_k(Q(k) \setminus X \cup Y) \) compared to each of the possible maximums for \( W(Y) - W(X) \).

If \( W(Y) - W(X) < 2[\alpha_k S - W(Q(k))] \)

\[
|r_k(Q(k) \setminus X \cup Y)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right|
= \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1
= \frac{[W(Y) - W(X)]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1
\]

Since \( W(Y) - W(X) < 2[\alpha_k S - W(Q(k))] \)

\[
< 2[\alpha_k S - W(Q(k))]P \frac{p(k)}{p(l)} + \frac{W(Q(k))P}{p(k)S} - 1
< - \frac{W(Q(k))P}{p(k)S} + 1
< - r_k(Q(k))
\]

Since \( r_A(Q(k)) < 0, -r_A(Q(k)) = |r_A(Q(k))| < |r_k(Q(k))| \)

\[
|r_k(Q(k) \setminus X \cup Y)| < |r_k(Q(k))|
\]

\[
|r_k(Q(k) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)
\]
If \( W(Y) - W(X) < W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)) \)

\[
|r_k(Q(k) \setminus X \cup Y)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right|
\]

\[
= \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1
\]

\[
= \frac{[W(Y) - W(X)]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1
\]

Since \( W(Y) - W(X) < W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)) \)

\[
< \frac{[W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k))]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1
\]

\[
< W(Q(l)) \left( \frac{p(k)}{p(l)} \right) \frac{P}{p(k)S} - \frac{W(Q(k))P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1
\]

\[
< \frac{W(Q(l))P}{p(l)S} - 1
\]

\[
< r_l(Q(l))
\]

\[
|r_k(Q(k) \setminus X \cup Y)| < |r_l(Q(l))|
\]

\[
\therefore |r_k(Q(k) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)
\]

Now to show that \( W(Y) - W(X) \geq \max \left( W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), 2[\alpha_k S - W(Q(k))] \right) \)

implies that \( |r_k(Q(k) \setminus X \cup Y)| \geq \max(|r_k(Q(k))|, |r_l(Q(l))|) \), consider \( |r_k(Q(k) \setminus X \cup Y)| \)
compared to both $|r_k(Q(k))|$ and $|r_l(Q(l))|$. 

\[ |r_k(Q(k) \setminus X \cup Y)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right| \]

\[ = \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right| = \left| \frac{[W(Y) - W(X)]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1 \right| \]

Since $W(Y) - W(X) \geq 2[\alpha_k S - W(Q(k))]$

\[ \geq \frac{2[\alpha_k S - W(Q(k))]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1 \]

\[ \geq \frac{2p(k)S - 2W(Q(k))P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1 \]

\[ \geq \frac{-W(Q(k))P}{p(k)S} + 1 \]

\[ \geq - r_k(Q(k)) \]

Since $r_k(Q(k)) \leq 0$, $-r_k(Q(k)) = |r_k(Q(k))|$

\[ \geq |r_k(Q(k))| \]

\[ |r_k(Q(k) \setminus X \cup Y)| \geq |r_k(Q(k))| \]
\[ |r_k(Q(k) \setminus X \cup Y)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right| \]
\[ = \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \]
\[ = \frac{[W(Y) - W(X)]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1 \]

Since \( W(Y) - W(X) \geq W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)) \)
\[ \geq \frac{[W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k))]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1 \]
\[ \geq W(Q(l)) \left( \frac{p(k)}{p(l)} \right) \frac{P}{p(k)S} - W(Q(k)) \frac{W(Q(k))P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1 \]
\[ \geq \frac{W(Q(l))P}{p(l)S} - 1 \]
\[ \geq r_l(Q(l)) \]

Since \( r_B(Q(l)) > 0, r_B(Q(l)) = |r_B(Q(l))| \)

\[ |r_k(Q(k) \setminus X \cup Y)| \geq |r_l(Q(l))| \]

\[ : |r_k(Q(k) \setminus X \cup Y)| \geq \max(|r_k(Q(k))|, |r_l(Q(l))|) \]

Thus \( \max(|r_k(Q(k) \setminus X \cup Y)|, |r_l(Q(l) \setminus X \cup Y)|) < \max(|r_k(Q(k))|, |r_l(Q(l))|) \) iff \( W(Y) - W(X) < \max \left( W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), 2[\alpha_k S - W(Q(k))] \right) \)

B.2.2 Case 2b

**Assumptions:** \( r_A(Q(k)) \leq 0 \) and \( r_B(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) \leq 0 \) and \( r_l(Q(l) \setminus X \cup Y) > 0 \).

By Lemma 3.3.20 \( |r_k(Q(k) \setminus X \cup Y)| < |r_k(Q(k))| \).

By Lemma 3.3.19 \( |r_l(Q(l) \setminus X \cup Y)| < |r_l(Q(l))| \).

Therefore, the error always decreases in this case.
B.2.3 Case 2c

Assumptions: \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus X \cup Y) \leq 0 \).

In this case the max error of the pair will be reduced iff

\[
W(Y) - W(X) < \max\left(W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)), 2[\alpha_k S - W(Q(k))]\right) \quad \text{and}
\]

\[
W(Y) - W(X) < \max\left(2W(Q(l)) - 2\alpha_l S, W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}\right).
\]

First consider the error of \( r_k(Q(k) \setminus X \cup Y) \).

If \( W(Y) - W(X) < 2[\alpha_k S - W(Q(k))] \)

\[
|r_k(Q(k) \setminus X \cup Y)| = \left|\frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1\right|
\]

\[
= \left|\frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1\right|
\]

Since \( W(Y) - W(X) < 2[\alpha_k S - W(Q(k))] \)

\[
< 2[\alpha_k S - W(Q(k))] \frac{P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1
\]

\[
< \frac{2p(k)S - 2W(Q(k))P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1
\]

\[
< \frac{-W(Q(k))P}{p(k)S} + 1
\]

\[
< - r_k(Q(k))
\]

Since \( r_k(Q(k)) \leq 0, -r_k(Q(k)) = |r_k(Q(k))| \)

\[
< |r_k(Q(k))|
\]

\[
|r_k(Q(k) \setminus X \cup Y)| < |r_k(Q(k))|
\]

\[
|r_k(Q(k) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)
\]
If $W(Y) - W(X) < W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k))$

$$|r_k(Q(k) \backslash X \cup Y)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right|$$

$$\frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1$$

$$\frac{[W(Y) - W(X)]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1$$

Since $W(Y) - W(X) < W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k))$

$$< \frac{[W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k))]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1$$

$$< W(Q(l)) \left( \frac{p(k)}{p(l)} \right) \frac{P}{p(k)S} - W(Q(k))P \frac{p(k)}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1$$

$$< \frac{W(Q(l))P}{p(l)S} - 1$$

$$< r_l(Q(l))$$

Since $r_B(Q(l)) > 0, r_B(Q(l)) > 0 = |r_l(Q(l))|$

$$|r_k(Q(k) \backslash X \cup Y)| < |r_l(Q(l))|$$

$$\therefore |r_k(Q(k) \backslash X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)$$
Now consider the error of $r_i(Q(l) \setminus X \cup Y)$. If $W(Y) - W(X) < 2W(Q(l)) - 2\alpha_i S$

$$|r_i(Q(l) \setminus X)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|$$

$$= -\left| \frac{[W(Q(l)) - (W(Y) - W(X))]P}{p(l)S} \right| + 1$$

$$= \left| \frac{(W(Y) - W(X))P}{p(l)S} \right| - \frac{W(Q(l))P}{p(l)S} + 1$$

$$= \left| \frac{(W(Y) - W(X))P}{p(l)S} \right| - \frac{W(Q(l))P}{p(l)S} + 1$$

Since $W(Y) - W(X) < 2W(Q(l)) - 2\alpha_i S$

$$< \frac{[2W(Q(l)) - 2\alpha_i S]P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1$$

$$< \frac{2W(Q(l))P}{p(l)S} - \frac{2p(l)S}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1$$

$$< \frac{W(Q(l))P}{p(l)S} - 2 + 1$$

$$< r_i(Q(l))$$

Since $r_B(Q(l)) > 0$, $r_B(Q(l)) > 0 = |r_i(Q(l))|$

$$|r_i(Q(l) \setminus Y \cup X)| < |r_i(Q(l))|$$
If \( W(Y) - W(X) < W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \)

\[
|r_l(Q(l) \setminus Y \cup X)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|
\]

\[
= -\frac{[W(Q(l)) - (W(Y) - W(X))]P}{p(l)S} + 1
\]

\[
= \frac{[(W(Y) - W(X))P}{p(l)S} - W(Q(l))P}{p(l)S} + 1
\]

\[
= \frac{[(W(Y) - W(X))P}{p(l)S} - W(Q(l))P}{p(l)S} + 1
\]

Since \( W(Y) - W(X) < W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \)

\[
< \frac{[W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}]P}{p(l)S} - W(Q(l))P}{p(l)S} + 1
\]

\[
< \frac{W(Q(l))P}{p(l)S} - W(Q(k)) \frac{p(l)}{p(k)} P}{p(l)S} - W(Q(l))P}{p(l)S} + 1
\]

\[
< - \frac{W(Q(k))P}{p(k)S} + 1
\]

\[
< - r_k(Q(k))
\]

Since \( r_A(Q(k)) \leq 0 \), and \( -r_A(Q(k)) = |r_A(Q(k))| \).

\[
|r_l(Q(l) \setminus Y \cup X)| < |r_k(Q(k))|
\]

\[
\therefore |r_l(Q(l) \setminus Y \cup X)| < \max(|r_k(Q(k))|, |r_l(Q(l))|). \text{ Thus}
\]

\[
\max(|r_k(Q(k)) \setminus X \cup Y|, |r_l(Q(l)) \setminus Y \cup X|) < \max(|r_k(Q(k))|, |r_l(Q(l))|).
\]

The error is decreased.

To show that if \( W(Y) - W(X) \geq \max \left( W(Q(l)) \frac{p(l)}{p(k)} - W(Q(k)), 2[\alpha_kS - W(Q(k))] \right) \) or \( W(Y) - W(X) \geq \max \left( 2W(Q(l)) - 2\alpha_lS, W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \right) \) again consider what happens in each case.
First consider the error of \( r_k(Q(k) \setminus X \cup Y) \). If \( W(Y) - W(X) \geq 2[\alpha_k S - W(Q(k))] \)

\[
|r_k(Q(k) \setminus X \cup Y)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right|
\]

\[
= \left( \frac{W(Q(k)) + W(Y) - W(X)}{p(k)S} - 1 \right) + \frac{W(Q(k))P}{p(k)S} - 1
\]

Since \( W(Y) - W(X) \geq 2[\alpha_k S - W(Q(k))] \)

\[
\geq \frac{2[\alpha_k S - W(Q(k))]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1
\]

\[
\geq \frac{2p(k)S - 2W(Q(k))P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1
\]

\[
\geq - \frac{W(Q(k))P}{p(k)S} + 1
\]

\[\geq - r_k(Q(k))\]

Since \( r_A(Q(k)) \leq 0 \), and \(-r_A(Q(k)) = |r_A(Q(k))|\).

\[\geq |r_k(Q(k))| \quad (r_k(Q(k)) < 0)\]

\[|r_k(Q(k) \setminus X \cup Y)| \geq |r_k(Q(k))|\]

\[|r_k(Q(k) \setminus X \cup Y)| \geq \max(|r_k(Q(k))|, |r_l(Q(l))|)\]
If \( W(Y) - W(X) \geq W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)) \)

\[
|r_k(Q(k) \setminus X \cup Y)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right| \\
= \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \\
= \frac{[W(Y) - W(X)]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1
\]

Since \( W(Y) - W(X) \geq W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)) \)
\[
\geq \frac{[W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k))]P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1 \\
\geq W(Q(l)) \left( \frac{p(k)}{p(l)} \right) \frac{P}{p(k)S} - W(Q(k)) \frac{P}{p(k)S} + \frac{W(Q(k))P}{p(k)S} - 1 \\
\geq \frac{W(Q(l))P}{p(l)S} - 1
\]

Since \( r_B(Q(l)) > 0, r_B(Q(l)) = |r_B(Q(l))| \)

\[
|r_k(Q(k) \setminus X \cup Y)| \geq |r_l(Q(l))| \\
\]
\[
\therefore |r_k(Q(k) \setminus X \cup Y)| \geq \max(|r_k(Q(k))|, |r_l(Q(l))|)
\]

Now consider the error of \( r_l(Q(l) \setminus X \cup Y) \).
If \( W(Y) - W(X) \geq 2W(Q(l)) - 2\alpha_t S \)

\[
|r_l(Q(l)\backslash Y \cup X)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]}{p(l)S} - 1 \right|
\]

\[
= -\frac{[W(Q(l)) - (W(Y) - W(X))]}{p(l)S} + 1
\]

\[
= \frac{[(W(Y) - W(X))]P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

Since \( W(Y) - W(X) \geq 2W(Q(l)) - 2\alpha_t S \)

\[
\geq \frac{[2W(Q(l)) - 2\alpha_t S]P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

\[
\geq \frac{2W(Q(l))P}{p(l)S} - \frac{2p(l)S}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

\[
\geq \frac{W(Q(l))P}{p(l)S} - 2 + 1
\]

\[
\geq r_l(Q(l))
\]

Since \( r_{\beta}(Q(l)) > 0, r_{\beta}(Q(l)) = |r_{\beta}(Q(l))| \)

\[
|r_l(Q(l)\backslash Y \cup X)| \geq |r_l(Q(l))|
\]
If \( W(Y) - W(X) \geq W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \),

\[
|r_i(Q(l) \setminus Y \cup X)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|
\]

\[
= -[\frac{W(Q(l)) - (W(Y) - W(X))P}{p(l)S}] + 1
\]

\[
= \frac{[(W(Y) - W(X))P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

\[
= \frac{[(W(Y) - W(X))P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

Since \( W(Y) - W(X) \geq W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \),

\[
\geq \frac{[W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}]P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

\[
\geq \frac{W(Q(k))P}{p(k)S} - \frac{W(Q(k))P}{p(k)} \frac{P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

\[
\geq - \frac{W(Q(k))P}{p(k)S} + 1
\]

\[
\geq - r_k(Q(k))
\]

Since \( r_A(Q(k)) \leq 0 \), \( -r_A(Q(k)) = |r_A(Q(k))| \)

\[
|r_i(Q(l) \setminus Y \cup X)| \geq |r_k(Q(k))|
\]

\[
\because |r_i(Q(l) \setminus Y \cup X)| \geq \max(|r_k(Q(k))|, |r_i(Q(l))|). \text{ Thus}
\]

\[
\max(|r_k(Q(k) \setminus X \cup Y)|, |r_i(Q(l) \setminus Y \cup X)|) \geq \max(|r_k(Q(k))|, |r_i(Q(l))|).
\]

The error does not decreases.

**B.2.4 Case 2d**

**Assumptions:** \( r_k(Q(k)) \leq 0 \) and \( r_i(Q(l)) > 0 \) and \( r_k(Q(k) \setminus X \cup Y) \leq 0 \) and \( r_i(Q(l) \setminus X \cup Y) \leq 0 \).

In this case the max error of the pair will be reduced iff \( W(Y) - W(X) < \max \left( 2W(Q(l)) - 2\alpha_iS, W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \right) \).

By Lemma 3.3.20 \( |r_k(Q(k) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_i(Q(l))|) \).
To show that $|r_{l}(Q\setminus X \cup Y)| < \max(|r_{k}(Q(k))|, |r_{l}(Q(l))|)$ consider $r_{l}(Q\setminus X \cup Y)$ when $W(Y) - W(X) < \max\left(2W(Q(l)) - 2\alpha_{l}S, W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}\right)$. If $W(Y) - W(X) < 2W(Q(l)) - 2\alpha_{l}S$

$$|r_{l}(Q\setminus Y \cup X)| = \left|\frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1\right|$$

$$= -\frac{[W(Q(l)) - (W(Y) - W(X))]P}{p(l)S} + 1$$

$$= \frac{[(W(Y) - W(X))]P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1$$

Since $W(Y) - W(X) < 2W(Q(l)) - 2\alpha_{l}S$

$$< \frac{2W(Q(l)) - 2\alpha_{l}S}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1$$

$$< \frac{2W(Q(l))P}{p(l)S} - \frac{2p(l)S}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1$$

$$< \frac{W(Q(l))P}{p(l)S} - 2 + 1$$

$$< r_{l}(Q(l))$$

Since $r_{l}(Q(l)) > 0$, $r_{B}(Q(l)) = |r_{B}(Q(l))|$

$$|r_{l}(Q\setminus Y \cup X)| < |r_{l}(Q(l))|$$
If \( W(Y) - W(X) < W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \)

\[
|r_i(Q(I) \cup Y \cup X)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|
\]

\[
= -\frac{[W(Q(l)) - (W(Y) - W(X))]P}{p(l)S} + 1
\]

\[
= \frac{[(W(Y) - W(X))P]{p(l)S} - W(Q(l))P}{p(l)S} + 1
\]

\[
= \frac{[(W(Y) - W(X))P]{p(l)S} - W(Q(l))P}{p(l)S} + 1
\]

Since \( W(Y) - W(X) < W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \)

\[
< \frac{[W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}P]{p(l)S} - W(Q(l))P}{p(l)S} + 1
\]

\[
< \frac{W(Q(l))P}{p(l)S} - W(Q(k)) \frac{p(l)}{p(k)P}{p(l)S} - W(Q(l))P}{p(l)S} + 1
\]

\[
< - \frac{W(Q(k))P}{p(k)S} + 1
\]

\[
< - r_k(Q(k))
\]

Since \( r_A(Q(k)) \leq 0, -r_A(Q(k)) = |r_A(Q(k))| \)

\[
|r_i(Q(I) \cup Y \cup X)| < |r_k(Q(k))|
\]

\[
\therefore |r_i(Q(I) \cup Y \cup X)| < \max(|r_k(Q(k))|, |r_i(Q(I))|). \text{ Thus in this case}
\]

\( W(Y) - W(X) < \max \left(2W(Q(l)) - 2\alpha l, W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \right) \) implies

\[
\max(|r_k(Q(k) \cup Y \cup X)|, |r_i(Q(I) \cup Y \cup X)|) < \max(|r_k(Q(k))|, |r_i(Q(l))|). \text{ Now to show that}
\]

the max error increase when \( W(Y) - W(X) \) does not meet those constraints

consider what happens to \( l \)’s error when \( W(Y) - W(X) \) is too large. If
\( W(Y) - W(X) \geq 2W(Q(l)) - 2\alpha_l S \)

\[
|r_l(Q(l) \setminus Y \cup X)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|
\]

\[
= -\left[ \frac{W(Q(l)) - (W(Y) - W(X))P}{p(l)S} + 1 \right]
\]

\[
= \left[ \frac{(W(Y) - W(X))P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1 \right]
\]

\[
= \left[ \frac{(W(Y) - W(X))P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1 \right]
\]

Since \( W(Y) - W(X) \geq 2W(Q(l)) - 2\alpha_l S \)

\[
\geq \frac{2[2W(Q(l)) - 2\alpha_l S]P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

\[
\geq \frac{2W(Q(l))P}{p(l)S} - \frac{2p(l)S}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

\[
\geq \frac{W(Q(l))P}{p(l)S} - 2 + 1
\]

\[
\geq r_l(Q(l))
\]

Since \( r_l(Q(l)) > 0, r_B(Q(l)) = |r_B(Q(l))| \)

\[
|r_l(Q(l) \setminus Y \cup X)| \geq |r_l(Q(l))|
\]
If $W(Y) - W(X) \geq W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}$

$$|r_I(Q(l)\setminus Y \cup X)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]}{p(l)S} - 1 \right|$$

$$= -\frac{[W(Q(l)) - (W(Y) - W(X))]P}{p(l)S} + 1$$

$$= \frac{[(W(Y) - W(X))P - W(Q(l))P}{p(l)S} + 1$$

Since $W(Y) - W(X) \geq W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}$

$$\geq \frac{W(Q(l))P}{p(l)S} - W(Q(k))\frac{p(l)}{p(k)} - \frac{W(Q(l))P}{p(l)S} + 1$$

$$\geq - \frac{W(Q(k))P}{p(k)S} + 1$$

$$\geq - r_k(Q(k))$$

Since $r_A(Q(k)) < 0$, $-r_A(Q(k)) = |r_A(Q(k))|$

$$|r_I(Q(l)\setminus Y \cup X)| \geq |r_k(Q(k))|$$

$\therefore |r_I(Q(l)\setminus Y \cup X)| \geq \max(|r_k(Q(k))|, |r_I(Q(l))|)$. Thus

$max(|r_k(Q(k))\setminus X \cup Y|, |r_I(Q(l)\setminus Y \cup X)|) \geq \max(|r_k(Q(k))|, |r_I(Q(l))|)$.

Therefore $W(Y) - W(X) \geq \max (2W(Q(l)) - 2\alpha_1S, W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)})$ implies

$max(|r_k(Q(k))\setminus X \cup Y|, |r_I(Q(l)\setminus Y \cup X)|) \geq \max(|r_k(Q(k))|, |r_I(Q(l))|)$.

### B.3 Case 3

#### B.3.1 Case 3a

$r_k(Q(k)) > 0$ and $r_I(Q(l)) > 0$ and $r_k(Q(k)\setminus X \cup Y) > 0$ and $r_I(Q(l)\setminus X \cup Y) > 0$.

In this case the error will be decreased if and only if

$W(Y) - W(X) < W(Q(l))\frac{p(k)}{p(l)} - W(Q(k))$. 

From the assumptions of this Theorem $r_A(Q(k)) < r_B(Q(l))$, and from the assumptions of this case $r_k(Q(k)) > 0$ and $r_l(Q(l)) > 0$. Thus $|r_A(Q(k))| < |r_B(Q(l))|$. By Lemma 3.3.19 $|r_l(Q(l) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)$. Therefore to show that the error reduces it is only necessary to show that $|r_k(Q(k) \setminus X \cup Y)| < |r_l(Q(l))|$.

$$|r_k(Q(k) \setminus X \cup Y)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]p}{p(k)S} - 1 \right|$$

Since $r_k(Q(k) \setminus X \cup Y) > 0$

$$= \frac{[W(Q(k)) + W(Y) - W(X)]p}{p(k)S} - 1$$

Since $W(Y) - W(X) < W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k))$

$$< \frac{[W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)) + W(Q(k))]p}{p(k)S} - 1$$

$$< W(Q(l)) \left( \frac{p(k)}{p(l)} \right) \frac{P}{p(k)S} - 1$$

$$< \frac{W(Q(l))P}{p(l)S} - 1$$

$$< r_l(Q(l))$$

Since $r_l(Q(l)) > 0$

$$|r_k(Q(k) \setminus X \cup Y)| < |r_l(Q(l))|$$

$\therefore |r_k(Q(k) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)$ The error is decreased.
To show that $W(Y) - W(X) \geq W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k))$ that will never decrease the error simply show that it will imply $|r_k(Q(k) \backslash X \cup Y)| \geq |r_l(Q(l))|$. 

\[
|r_k(Q(k) \backslash X \cup Y)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]^p}{p(k)S} - 1 \right|
\]

Since $r_k(Q(k) \backslash X \cup Y) > 0$

\[
= \frac{[W(Q(k)) + W(Y) - W(X)]^p}{p(k)S} - 1
\]

Since $W(Y) - W(X) \geq W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k))$

\[
\geq \frac{[W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k)) + W(Q(k))]^p}{p(k)S} - 1
\]

\[
\geq W(Q(l)) \left( \frac{p(k)}{p(l)} \right) \frac{P}{p(k)S} - 1
\]

\[
\geq W(Q(l)) \frac{P}{p(l)S} - 1
\]

\[
\geq r_l(Q(l))
\]

Since $r_l(Q(l)) > 0$

\[
|r_k(Q(k) \backslash X \cup Y)| \geq |r_l(Q(l))|
\]

\[\therefore |r_k(Q(k) \backslash X \cup Y)| \geq \max(|r_k(Q(k))|, |r_l(Q(l))|) \text{ The error is increased.}\]

**B.3.2 Case 3b,d**

**Assumptions:** $r_k(Q(k)) > 0$ and $r_l(Q(l)) > 0$ and $r_k(Q(k) \backslash X \cup Y) \leq 0$.

The case is impossible as it would imply that moving more weight to processor $k$ would actually decrease its load. The following demonstrates that
$r_k(Q(k) \setminus X \cup Y) > 0$ must be true.

$$r_k(Q(k) \setminus X \cup Y) = \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1$$

$$= \frac{W(Q(k))P}{p(k)S} - 1 + [W(Y) - W(X)] \frac{P}{p(k)} S$$

$$= r_k(Q(k)) + [W(Y) - W(X)] \frac{P}{p(k)} S$$

Since $r_k(Q(k)) > 0$

$$> 0 + [W(Y) - W(X)] \frac{P}{p(k)} S$$

Since $W(Y) - W(X) > 0$

$$> 0 + [0] \frac{P}{p(k)} S$$

$$r_k(Q(k) \setminus X \cup Y) > 0$$

Thus, this case cannot happen.

**B.3.3 Case 3c**

**Assumptions:** $r_k(Q(k)) > 0$ and $r_l(Q(l)) > 0$ and $r_k(Q(k) \setminus X \cup Y) > 0$ and $r_l(Q(l) \setminus X \cup Y) \leq 0$. In this case the error is reduced iff

$W(Y) - W(X) < 2W(Q(l)) - 2\alpha_l S$ and $W(Y) - W(X) < W(Q(l)) \frac{p(k)}{p(l)} - W(Q(k))$.

From the assumptions of this Theorem $r_A(Q(k)) < r_B(Q(l))$, and from the assumptions of this case $r_k(Q(k)) > 0$ and $r_l(Q(l)) > 0$. Thus $|r_A(Q(k))| < |r_B(Q(l))|$. Therefore it is only necessary to show that $|r_l(Q(l) \setminus Y \cup X)| < |r_l(Q(l))|$ and $r_k(Q(k) \setminus X \cup Y) < |r_l(Q(l))|$. 
Showing that $r_k(Q(k) \setminus X \cup Y) < |r_l(Q(l))|$ is similar to case 3a.

$$|r_k(Q(k) \setminus X \cup Y)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]p}{p(k)S} - 1 \right|$$

Since $r_k(Q(k) \setminus X \cup Y) > 0$

$$= \frac{[W(Q(k)) + W(Y) - W(X)]p}{p(k)S} - 1$$

Since $W(Y) - W(X) < W(Q(l))p(k) - W(Q(k))$

$$< \frac{[W(Q(l))p(k) - W(Q(k)) + W(Q(k))]p}{p(k)S} - 1$$

$$< \frac{W(Q(l))p(k)}{p(l)S} - 1$$

$$< r_l(Q(l))$$

Since $r_l(Q(l)) > 0$

$$|r_k(Q(k) \setminus X \cup Y)| < |r_l(Q(l))|$$

$$\therefore |r_k(Q(k) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)$$
The following demonstrates that $|r_l(Q(l) \setminus Y \cup X)| < |r_l(Q(l))|$ it true.

$$|r_l(Q(l) \setminus Y \cup X)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|$$

$$= -\frac{[W(Q(l)) - (W(Y) - W(X))]P}{p(l)S} + 1$$

$$= -\frac{[(W(Y) - W(X))P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1}$$

$$= -\frac{[(W(Y) - W(X)P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1}$$

Since $W(Y) - W(X) < 2W(Q(l)) - 2\alpha_l S$

$$<\frac{[2W(Q(l)) - 2\alpha_l S]P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1$$

$$<\frac{2W(Q(l))P}{p(l)S} - \frac{2p(l)S}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1$$

$$<\frac{W(Q(l))P}{p(l)S} - 2 + 1$$

Since $r_l(Q(l)) > 0$

$$<r_l(Q(l))$$

$$|r_l(Q(l) \setminus Y \cup X)| < |r_l(Q(l))|$$

Thus $|r_l(Q(l) \setminus Y \cup X)| < |r_l(Q(l))|$ and $|r_k(Q(k) \setminus X \cup Y)| < |r_l(Q(l))|$ which implies $\max(|r_l(Q(l) \setminus Y \cup X)|, |r_k(Q(k) \setminus X \cup Y)|) < \max(|r_k(Q(k))|, |r_l(Q(l))|)$. The error is always decreased.
To show that \( W(Y) - W(X) \geq 2W(Q(l)) - 2\alpha_l S \) will never decrease the error examine the effect it has on \( |r_l(Q(l) \backslash Y \cup X)| \).

\[
|r_l(Q(l) \backslash Y \cup X)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|
\]

\[
= \frac{-[W(Q(l)) - (W(Y) - W(X))]P}{p(l)S} + 1
\]

\[
= \frac{[(W(Y) - W(X))]P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

Since \( W(Y) - W(X) \geq 2W(Q(l)) - 2\alpha_l S \)

\[
\geq \frac{[2W(Q(l)) - 2\alpha_l S]P}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

\[
\geq \frac{2W(Q(l))P}{p(l)S} - \frac{2p(l)S}{p(l)S} - \frac{W(Q(l))P}{p(l)S} + 1
\]

\[
\geq \frac{W(Q(l))P}{p(l)S} - 2 + 1
\]

Since \( r_l(Q(l)) > 0 \)

\[
\geq r_l(Q(l))
\]

\[
|r_l(Q(l) \backslash Y \cup X)| \geq |r_l(Q(l))|
\]

\[
|r_l(Q(l) \backslash Y \cup X)| \geq \max(|r_k(Q(k))|, |r_l(Q(l))|)
\]

Thus the error will never decrease.

**B.4 Case 4**

**B.4.1 Case 4a,b**

**Assumptions:** \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) \leq 0 \) and \( r_l(Q(l) \backslash Y \cup X) > 0 \).

This is another impossible case, since it would imply that removing weight from a processor was increasing its load. The following demonstrates that
\( r_l(l \setminus Y \cup X) < 0 \) must be true if \( r_l(Q(l)) < 0 \) and \( W(Y) - W(X) > 0 \).

\[
\begin{align*}
r_l(Q(l) \setminus Y \cup X) &= \frac{[W(Q(l)) - W(Y) + W(X)]p}{p(l)S} - 1 \\
&= \frac{W(Q(l))p}{p(l)S} - 1 - \frac{W(Y) - W(X)}{p(l)S} \\
&= r_l(Q(l)) - \frac{W(Y) - W(X)}{p(l)S}
\end{align*}
\]

Since \( r_l(Q(l)) < 0 \)

\[
r_l(Q(l) \setminus Y \cup X) < 0 - \frac{W(Y) - W(X)}{p(l)S}
\]
Since \( W(Y) - W(X) > 0 \)

\[
r_l(Q(l) \setminus Y \cup X) < 0 - 0r_l(Q(l) \setminus Y \cup X) < 0
\]
Therefore, this case cannot happen.

**B.4.2 Case 4c**

**Assumptions:** \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) \leq 0 \) and \( r_k(Q(k) \setminus X \cup Y) > 0 \) and \( r_l(Q(l) \setminus X \cup Y) \leq 0 \).

The error in this case will decrease if and only if

\[
W(Y) - W(X) < W(Q(l)) - W(Q(k)) \cdot \frac{p(l)}{p(k)} \quad \text{and} \quad W(Y) - W(X) < 2\alpha_k S - 2W(Q(k)).
\]

From the assumptions of this Theorem \( r_A(Q(k)) < r_B(Q(l)) \), and from the assumptions of this case \( r_k(Q(k)) \leq 0 \) and \( r_l(Q(l)) \leq 0 \). Thus \( |r_A(Q(k))| \geq |r_B(Q(l))| \).

Therefore it is only necessary to show that \( r_k(Q(k) \setminus X \cup Y) < |r_A(Q(k))| \) and that \( r_l(Q(l) \setminus X \cup Y) < |r_A(Q(k))| \).
Starting with $|r_k(Q(k) \setminus X \cup Y)| < |r_k(Q(k))|$.  

$$
|r_k(Q(k) \setminus X \cup Y)| = \left| \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1 \right|
$$

$$
= \frac{[W(Q(k)) + W(Y) - W(X)]P}{p(k)S} - 1
$$

$$
= \frac{W(Q(k))P}{p(k)S} + \frac{[W(Y) - W(X)]P}{p(k)S} - 1
$$

Since $W(Y) - W(X) < 2\alpha_k S - 2W(Q(k))$

$$
< \frac{W(Q(k))P}{p(k)S} + \frac{[2\alpha_k S - 2W(Q(k))]P}{p(k)S} - 1
$$

$$
< \frac{W(Q(k))P}{p(k)S} + \frac{2P(k)S - 2W(Q(k))P}{p(k)S} - 1
$$

$$
< - \frac{2W(Q(k))P}{p(k)S} - 1 + 2
$$

$$
< - r_k(Q(k))
$$

Since $r_k(Q(k)) \leq 0$

$$
|r_k(Q(k) \setminus X \cup Y)| < |r_k(Q(k))|
$$

$$
|r_k(Q(k) \setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)
$$
The following shows that $|r_l(Q(l)\setminus X \cup Y)| < |r_k(Q(k))|$. 

$$|r_l(Q(l)\setminus X \cup Y)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|$$

$$= \frac{[-W(Q(l)) + W(Y) - W(X)]P}{p(l)S} + 1$$

Since $W(Y) - W(X) < W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}$

$$\leq \frac{[-W(Q(l)) + W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}]P}{p(l)S} + 1$$

$$\leq \frac{-W(Q(k))P}{p(k)S} + 1$$

$$< - r_k(Q(k))$$

Since $r_k(Q(k)) \leq 0$

$$|r_l(Q(l)\setminus X \cup Y)| < |r_k(Q(k))|$$

$$|r_l(Q(l)\setminus X \cup Y)| < \max(|r_k(Q(k))|, |r_l(Q(l))|)$$

$\therefore \max(|r_k(Q(k)\setminus X \cup Y)|, |r_l(Q(l)\setminus X \cup Y)|) < \max(|r_k(Q(k))|, |r_l(Q(l))|)$.

The error will decreases.

To show that the conditions are necessary examine what happens when

$W(Y) - W(X) \geq W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}$ or $W(Y) - W(X) \geq 2\alpha k S - 2W(Q(k))$. 
If $W(Y) - W(X) \geq 2\alpha_k S - 2W(Q(k))$.

$$|r_k(Q(k) \setminus X \cup Y)| = \left|\frac{[W(Q(k)) + W(Y) - W(X)]}{p(k)S} - 1\right|$$

$$= \frac{W(Q(k))P}{p(k)S} + \frac{[W(Y) - W(X)]}{p(k)S} - 1$$

Since $W(Y) - W(X) \geq 2\alpha_k S - 2W(Q(k))$

$$\geq \frac{W(Q(k))P}{p(k)S} + \frac{2\alpha_k S - 2W(Q(k))}{p(k)S} - 1$$

$$\geq \frac{W(Q(k))P}{p(k)S} - \frac{2W(Q(k))}{p(k)S} - 1$$

$$\geq - \frac{2W(Q(k))P}{p(k)S} - 1 + 2$$

$$\geq - r_k(Q(k))$$

Since $r_k(Q(k)) \leq 0$

$$|r_k(Q(k) \setminus X \cup Y)| \geq |r_k(Q(k))|$$

$$|r_k(Q(k) \setminus X \cup Y)| \geq \max(|r_k(Q(k))|, |r_l(Q(l))|)$$
If $W(Y) - W(X) \geq W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}$,

$$|r_i(Q(l) \setminus X \cup Y)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]P}{p(l)S} - 1 \right|$$

$$= \left| \frac{[-W(Q(l)) + W(Y) - W(X)]P}{p(l)S} + 1 \right|$$

Since $W(Y) - W(X) \geq W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}$

$$\geq \frac{[-W(Q(k))P}{p(l)S} \left( \frac{p(l)}{p(k)} \right) + 1$$

$$\geq \frac{[-W(Q(k))P}{p(k)S} + 1$$

$$\geq - r_k(Q(k))$$

Since $r_k(Q(k)) \leq 0$

$$|r_i(Q(l) \setminus X \cup Y)| \geq |r_k(Q(k))|$$

$$|r_i(Q(l) \setminus X \cup Y)| \geq \max(|r_k(Q(k))|, |r_i(Q(l))|)$$

$$\therefore \max(|r_k(Q(k)) \setminus X \cup Y|, |r_i(Q(l)) \setminus X \cup Y|) \geq \max(|r_k(Q(k))|, |r_i(Q(l))|).$$

The error will never decrease.

Thus $\max(|r_k(Q(k)) \setminus X \cup Y|, |r_i(Q(l)) \setminus X \cup Y|) \geq \max(|r_k(Q(k))|, |r_i(Q(l))|)$ if and only if $W(Y) - W(X) < W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}$ and $W(Y) - W(X) < 2\alpha_k S - 2W(Q(k))$.

**B.4.3 Case 4d**

**Assumptions:** $r_k(Q(k)) \leq 0$ and $r_i(Q(l)) \leq 0$ and $r_k(Q(k) \setminus X \cup Y) > 0$ and $r_i(Q(l) \setminus X \cup Y) \leq 0$.

In this case the error will decrease iff $W(Y) - W(X) < W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)}$.

Again note the $W(Q(l)) - W(Q(k))\frac{p(l)}{p(k)} > 0$ must be true and that it implies that $|r_k(Q(k))| > |r_i(Q(l))|$ just as it did in case 4c.

By Lemma 3.3.20 $|r_k(Q(k) \setminus X \cup Y| < \max(|r_k(Q(k))|, |r_i(Q(l))|)$. 
Now to show that $|r_l(Q(l) \setminus X \cup Y)| < |r_k(Q(k))|$. 

$$|r_l(Q(l) \setminus X \cup Y)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]p}{p(l)S} - 1 \right|$$

$$= \frac{[-W(Q(l)) + W(Y) - W(X)]p}{p(l)S} + 1$$

Since $W(Y) - W(X) < W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}$

$$< \frac{[-W(Q(l)) + W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}]p}{p(l)S} + 1$$

$$< \frac{-W(Q(k))p}{p(l)S} \left( \frac{p(l)}{p(k)} \right) + 1$$

$$< \frac{-W(Q(k))p}{p(k)S} + 1$$

$$< - r_k(Q(k))$$

Since $r_k(Q(k)) \leq 0$

$$|r_l(Q(l) \setminus X \cup Y)| < |r_k(Q(k))|$$

$$|r_l(Q(l) \setminus X \cup Y)| < \max(|r_k(Q(k)|, |r_l(Q(l))|)$$

\[ \therefore \max(|r_k(Q(k) \setminus X \cup Y)|, |r_l(Q(l) \setminus X \cup Y)|) < \max(|r_k(Q(k)|, |r_l(Q(l))|). \]

The error decreases. Next show that $W(Y) - W(X) \geq W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}$ will
increase the error.

\[
|r_l(Q(l) \setminus X \cup Y)| = \left| \frac{[W(Q(l)) - W(Y) + W(X)]}{p(l)S} - 1 \right|
\]

\[
= \left| \frac{[-W(Q(l)) + W(Y) - W(X)]}{p(l)S} + 1 \right|
\]

Since \( W(Y) - W(X) \geq W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \)

\[
\geq \left| \frac{[-W(Q(l)) + W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)}]}{p(l)S} + 1 \right|
\]

\[
\geq \left| \frac{-W(Q(k))}{p(k)S} \right| + 1
\]

\[
\geq - r_k(Q(k))
\]

Since \( r_k(Q(k)) \leq 0 \)

\[
|r_l(Q(l) \setminus X \cup Y)| \geq |r_k(Q(k))|
\]

\[
|r_l(Q(l) \setminus X \cup Y)| \geq \max(|r_k(Q(k))|, |r_l(Q(l))|)
\]

\[
\therefore \max(|r_k(Q(k) \setminus X \cup Y)|, |r_l(Q(l) \setminus X \cup Y)|) \geq \max(|r_k(Q(k))|, |r_l(Q(l))|).
\]

The error increases.

The error will decrease if \( W(Y) - W(X) < W(Q(l)) - W(Q(k)) \frac{p(l)}{p(k)} \).