Effective Capacity Planning of the Virtual Environment using Enterprise Architecture

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Arati Mahimane

Graduate Program in Computer Science and Engineering

The Ohio State University

2013

Master's Examination Committee:

Professor Jayashree Ramanathan, Advisor

Professor Rajiv Ramnath
Abstract

Capacity planning is an uphill task for any enterprise to estimate their future demands and it is really difficult to plan out the production capacity that an enterprise requires for its applications. The main challenge is to predict the demand of resources to meet the Service Level Agreements on an ongoing basis. Underestimation of resources can cause resource shortage and consequent revenue loss. Overestimation can lead to idle resources and increased shared costs. It is necessary to utilize these resources properly in order to decrease cost. The final goal of our research is to predict, prepare and measure the use of resources as close to the actual demand as possible.

Our research proposes an innovative concept of using Enterprise Architecture for effective capacity planning. We aggregate application usage statistics from the physical environment and then predict the capacity requirements for a virtual environment. We show how the relationships and data in an Enterprise Architecture repository can be used to predict the capacity and deployment for an application. The dependencies within the components of an enterprise play a major role in the prediction process. Our research helps in eliminating the problem of over provisioning by maximizing the utilization of available resources. We show that by using our approach the memory utilization of a virtual machine increases by 40-50% and CPU utilization increases by 8-12%.
Dedication

This document is dedicated to my family.
Acknowledgments

I would like to thank Dr. Jay Ramanathan for her motivation, support, guidance and patience throughout the work in this thesis. Thanks to Dr. Rajiv Ramnath for guiding me, especially during the initial phases of my Masters.

I also want to thank Zahid Hossain for his continued support and training in Enterprise Architecture.

I thank the capacity planning and application teams at Nationwide Insurance for providing me the data and other information.

I would like to thank all my friends, especially Abhishek, Ramya and Shweta for their encouragement and help.

I would like to mention a special thanks to my fiancé Siddharth Mulay for being a patient listener and for providing constant help and support throughout my Masters.

Finally I would like to thank my parents without whom I would not have come all this way. I am forever indebted to them for everything they have done for me.
Vita

2005 to 2009 ................................ Bachelor of Engineering (B.E.) in Computer Engineering, Cummins College of Engineering for Women, Pune, India.

Aug 2009 to July 2011 ....................... Software Engineer, Persistent Systems Ltd.

Pune, India.

June 2012 to Aug 2012 ....................... IT Intern, Nationwide Insurance,

Columbus, Ohio, USA.

Aug 2012 to present .......................... Graduate Research Associate, Department of Computer Science and Engineering, The Ohio State University, Columbus, Ohio, USA.

Fields of Study

Major Field: Computer Science and Engineering
Table of Contents

Abstract ................................................................................................................................. ii

Acknowledgments .............................................................................................................. iv

Vita ......................................................................................................................................... v

List of Tables ......................................................................................................................... ix

List of Figures ...................................................................................................................... x

Chapter 1: Introduction ......................................................................................................... 1

1.1 Innovation ....................................................................................................................... 2

1.2 Hypothesis ....................................................................................................................... 2

1.3 Motivation ....................................................................................................................... 2

1.4 Research Contributions ................................................................................................. 3

1.5 Thesis Outline ............................................................................................................... 4

Chapter 2: Enterprise Architecture ...................................................................................... 5

2.1 What is Enterprise Architecture? .................................................................................. 5

2.2 Domains of Enterprise Architecture ............................................................................. 6

2.3 Outcome of a good EA practice .................................................................................... 7

2.4 Nationwide’s Enterprise Architecture Content Metamodel .......................................... 9

vi
Chapter 3: Capacity Planning ................................................................. 14

3.1 What is Capacity Planning? ............................................................. 14

3.2 Why is it important? ....................................................................... 15

3.3 How Capacity planning can help achieve EA goals ......................... 16

Chapter 4: Related Work ..................................................................... 18

Chapter 5: Case Study - Nationwide ...................................................... 23

5.1 Enterprise Architecture AS-IS Scenario ........................................ 23

5.2 Elements missing from Enterprise Architecture ............................. 29

5.3 Capacity Planning AS-IS Scenario ............................................... 31

5.4 Problem of Over-provisioning ......................................................... 32

Chapter 6: Research and Contributions ................................................. 34

6.1 Problem Analysis ........................................................................... 34

6.2 Goals and Methodology .................................................................. 35

6.3 System Architecture ...................................................................... 36

6.4 Parameters required for effective Capacity Planning ...................... 38

6.5 Algorithm for Capacity Planning .................................................. 40

6.6 Challenges ....................................................................................... 43

6.7 Data Collection ............................................................................... 43

6.8 Results and Validation .................................................................... 46

vii
Chapter 7: Business Value ........................................................................................................ 50

  7.1 Service Blueprint ........................................................................................................ 50

  7.2 Benefits to the Java Hosting team ............................................................................ 52

  7.3 Business value at each level in the Enterprise Architecture ............................... 53

Chapter 8: Conclusion and Future Work ........................................................................ 55

References ............................................................................................................................. 56

APPENDIX A: Details of the algorithm ........................................................................... 58

APPENDIX B: Java Implementation .................................................................................. 61
List of Tables

Table 1: Application Components ................................................................. 44
Table 2: Infrastructure Components (Virtual Machines) .............................. 45
List of Figures

Figure 1: Enterprise Architecture................................................................. 6
Figure 2: Domains of Enterprise Architecture.................................................... 7
Figure 3: Nationwide's Enterprise Architecture Content Metamodel 2.2............... 13
Figure 4: Application Component - Introduction tab ...................................... 24
Figure 5: Application Component - Decomposition tab.................................... 25
Figure 6: Application Component - Operational tab ........................................ 25
Figure 7: Application Component - Service tab .............................................. 26
Figure 8: Application Component - Technology tab......................................... 26
Figure 9: Infrastructure Component - Introduction tab................................. 27
Figure 10: Infrastructure Component - Operational tab .................................. 28
Figure 11: Infrastructure Component - Application tab .................................. 29
Figure 12: Over provisioning - Low CPU Utilization ....................................... 32
Figure 13: Over provisioning - Low Memory Utilization .................................... 33
Figure 14: Goals.............................................................................................. 35
Figure 15: System Architecture ....................................................................... 36
Figure 16: Parameters required for each Application Component .................. 39
Figure 17: Parameters required for each Infrastructure Component ................ 40
Figure 18: Capacity Planning Algorithm........................................................ 42
Figure 19: Results - Memory Utilization a) AS-IS b) TO-BE............................ 47
Figure 20: Results - CPU Utilization a) AS-IS b) TO-BE..................................................48
Figure 21: Service Blueprint................................................................................................51
Figure 22: Business Value .....................................................................................................54
Chapter 1: Introduction

This chapter gives a brief introduction about Enterprise Architecture and Capacity Planning. It also talks about our innovation, hypothesis, motivation towards this research and research contributions.

Enterprise Architecture is a blueprint of an enterprise. It is a coherent description of the structure of products, services, processes, organization, data, applications and technology, including underlying assumptions and principles. It helps in aggregating all information in one central architectural repository and gain useful insights from this information.

Capacity planning is the process of predicting the production capacity that an enterprise requires for its applications. Underestimation of resources can cause resource shortage and consequent revenue loss. Overestimation can lead to idle resources and increased shared costs. The final goal of our research is to predict, prepare and measure the use of resources as close to the actual demand as possible.

Our research embraces the concept of using Enterprise Architecture for improving the process of capacity planning.
1.1 Innovation

To build a comprehensive Framework that leverages Enterprise Architecture to improve the process of Capacity Planning for an Insurance Enterprise.

1.2 Hypothesis

If the Enterprise Architecture framework is enhanced with application operational data from the physical environment, then it helps in effective capacity planning of the virtual environment.

1.3 Motivation

The ITIL library describes Capacity Management as a balancing act, where we try to balance cost against capacity and supply against demand. Capacity Planning is a key step in Business-IT Alignment. Cloud-based storage and computing are growing in popularity due to economies of using a centralized infrastructure that can be leased at low cost. Cloud computing actually adds complexity to the capacity management problem. The increased use of virtual-machine-based server consolidation in such data centers introduces new challenges for resource management, capacity provisioning, and guaranteeing application performance. Providing capacity based on the peak rate can result in significant over-provisioning and low utilization, leading to higher infrastructure
and energy costs. Hence, capacity planning of the virtual environments is of crucial importance to every enterprise.

At Nationwide, we have found out problems of over provisioning of capacity and the planning process also requires several manual steps. We also wanted to aggregate all the information required for capacity planning in one central location hence we have chosen to use Enterprise Architecture repository for doing this.

1.4 Research Contributions

Our research proposes an algorithm for automating the capacity planning process. The algorithm helps in accurate prediction of application capacity requirements for efficient resource utilization on virtual environments. We show how the relationships and data from Enterprise Architecture repository can play a major role in the capacity planning process. Our research also helps identify the missing components and thereby facilitates the enhancement of the Enterprise Architecture framework for effective capacity planning. We have validated our algorithm over a dataset consisting of 8 applications and 3 virtual machines.
1.5 Thesis Outline

This work describes an algorithm for capacity planning and how this can be incorporated into Enterprise Architecture. The rest of this thesis is broken down into following chapters.

Chapter two describes Enterprise Architecture and its goals. Chapter three describes what capacity planning is and why it is important for enterprises. Chapter four talks about the related work and gives a brief description about various related topics. Chapter five discusses the case study and the findings from it. Chapter six describes the research done and the contributions of this work towards the capacity planning problem. It discusses the approach that should be used to improve the process of capacity planning.

Chapter seven discusses the effectiveness of this work and finally chapter eight concludes our work and talks about the future work that needs to be done in this area.
Chapter 2: Enterprise Architecture

This chapter describes Enterprise Architecture, its domains and goals. It also describes Nationwide’s Enterprise Architecture Content Metamodel in detail.

2.1 What is Enterprise Architecture?

*Enterprise architecture is an ongoing business function that helps an enterprise figure out how to execute best the strategies that drive its development.*

Enterprise Architecture is a blueprint which acts as a collaboration force among

- aspects of business planning (strategic) such as goals, visions, strategies, capabilities and governance principles
- aspects of business operations such as business terms, organization structures, processes, and data (information)
- aspects of automation such as information systems and databases
- aspects of the enabling technological infrastructure of the business such as computers, operating systems and networks

Figure 1 shows the levels in an Enterprise Architecture.
2.2 Domains of Enterprise Architecture

Figure 2 shows the domains of an Enterprise Architecture. These are discussed below:

**Business Architecture:** It consists of business strategy, governance, organization, and key business processes.

**Service Architecture:** It consists of business capability to be deployed, their interactions, and relationships to other enterprise entities such as business process and data entities.

**Data Architecture:** It consists of structure of an organization’s logical and physical data assets and data management resources.
**Application Architecture:** It consists of application systems to be deployed and their relationships to other enterprise entities such as technology and infrastructure components.

**Technology Architecture:** It consists of software and hardware capabilities that are required to support the deployment of business, data, and applications.

---

![Diagram: Domains of Enterprise Architecture](image)

**Figure 2:** Domains of Enterprise Architecture

---

### 2.3 Outcome of a good EA practice

Enterprise Architecture helps in achieving the following:

- A more efficient business operation
  - Lower business operation costs
- More agile organization
- Business capabilities shared across the organization
- Lower Change Management Costs
- More flexible workforce
- Improved business productivity

• A more efficient IT operation
  - Lower software development, support, and maintain costs
  - Increased portability of applications
  - Improved interoperability and easier system and network management
  - Improved ability to address critical enterprise-wide issues, such as security
  - Easier upgrade and exchange of system components

• Better return on existing investment, reduced risk for future investment
  - Reduced complexity in the business and IT
  - Maximum return on investment in existing business and IT infrastructure
  - The flexibility to make, buy, or outsource business or IT solutions
  - Reduced risk overall in new investments and their cost of ownership

• Faster, simpler, and cheaper procurement
  - Simpler buying decisions, because the information governing procurement is readily available in a coherent plan
  - Faster procurement process, maximizing procurement speed and flexibility without sacrificing architectural coherence
Nationwide's EA Vision and Mission

**Vision:** Enterprise architecture will enable Nationwide to plan better and react faster to business and technology events and run an effective and efficient organization.

**Mission:** Create a common understanding of dependencies between Nationwide’s business and IT architectural building blocks (elements) with a consistent, coherent, and repeatable process and taxonomy.

### 2.4 Nationwide's Enterprise Architecture Content Metamodel

TOGAF®, an Open Group Standard, is a proven enterprise architecture methodology and framework used by the world's leading organizations to improve business efficiency. It is the most prominent and reliable enterprise architecture standard, ensuring consistent standards, methods, and communication among enterprise architecture professionals.

Nationwide's content metamodel is based on TOGAF 9 Framework.

The content metamodel provides a definition of all the types of building blocks that may exist within an architecture, showing how these building blocks can be described and related to one another.

Example: when creating an architecture, an architect will identify applications, “data entities” held within applications, and technologies that implement those applications. These applications will in turn support particular groups of business user or actor, and will be used to fulfill “business services”.

9
The content metamodel identifies all of these concerns (i.e., application, data entity, technology, actor, and business service), shows the relationships that are possible between them (e.g., actors consume business services), and finally identifies artifacts that can be used to represent them.

As shown in Figure 3 we discuss the building blocks of the content metamodel below:

**Business Architecture Building Blocks**

- **Function** delivers business capabilities closely aligned to an organization, but not necessarily explicitly governed by the organization. Also referred to as "business function". E.g. Sales, Pricing & Underwriting, CRM.

- A **Process** represents flow of control between or within functions and/or services. Processes represent a sequence of activities that together achieve a specified outcome, can be decomposed into sub-processes.

- A process is triggered by an **Event**.

- An **Actor** is a person, organization, system or stakeholder that participates in the activities of a process and performs a role using specific competencies. An actor can generate events and can consume products, services and information. An actor can play different roles as part of processes.

- **Product** is an output generated by the business.
• **Business Service** supports business capabilities through an explicitly defined interface and is explicitly governed by an organization. E.g. Take a Payment, Re-instatement policy.

• **Information System (IS) Service** represents the automated elements of a business service. An information system service may deliver or support part or all of one or more business services E.g. Process Payment, Amend Policy.

**Data Architecture Building Blocks**

• **Data Entity** is an encapsulation of data that is recognized by a business domain expert as a thing. Logical data entities can be tied to repositories and services and may be structured according to implementation considerations.

• **Datastore** is a database that is managed and governed by an organization

• **Reference Data Entity** is a Data Entity as defined in the Reference Data Models adopted by Nationwide.

**Service Architecture Building Blocks**

• **Information System (IS) Service** represents the automated elements of a business service. An Information System (IS) Service may deliver or support part or all of one or more business services, however, for our purposes, an IS Services is viewed as an automation of business service. A business service may have 0 or more IS Service implementations.
• **Mediator** is a special role an application component plays in the invocation process of a service. The Mediator does not change the functional behavior of the service but decouples the provider from the consumer and may provide protocol mediation, message transformation, routing, and security.

**Application Architecture Building Blocks**

• **Application Component** is an encapsulation of application functionality aligned to implementation structure. At the lowest level an application component is an independently deployable software component and provides service(s). Example: a purchase request processing application.

• **Mediator** (described above) is a part of Service Architecture as well as Application Architecture.

**Technology Architecture Building Blocks**

• **Technology Component** is an encapsulation of technology infrastructure that represents a class of technology product or a specific technology product.

• **Infrastructure Component** is an (physical) instance of a technology component, such as servers, network, and storage units.

• **Infrastructure Solution Pattern** is a logical group of related Technology Components frequently used together to implement a solution.
Figure 3: Nationwide's Enterprise Architecture Content Metamodel 2.2
Chapter 3: Capacity Planning

This chapter describes what capacity planning is and why it is important for enterprises. It also talks about how capacity planning can help achieve goals of Enterprise Architecture.

3.1 What is Capacity Planning?

As its name implies, the systems management discipline of capacity planning involves the planning of various kinds of resource capacities for an infrastructure.

*Capacity planning is a process to predict the types, quantities, and timing of critical resource capacities that are needed within an infrastructure to meet accurately forecasted workloads.*

Effective capacity planning makes efficient use of existing capacity, minimizes waste of computing resources, and plans for the need to accommodate any spikes in resource demands. To address the challenges of managing capacity in the virtualized data center,
IT managers need end-to-end visibility into their physical and virtual resources. Understanding, forecasting, and fine tuning application and infrastructure component usage helps to improve performance and reduce consumption, which lowers costs and allows more consistent levels of IT services. With an effective capacity planning strategy in place, IT operations can provide more efficient and comprehensive service levels, while making efficient use of the existing infrastructure.

Following are the steps typically followed to develop an effective capacity planning process:

i. Identify the key resources to be measured.

ii. Measure the utilizations or performance of the resources.

iii. Compare utilizations to maximum capacities.

iv. Collect workload forecasts from developers and users.

v. Transform workload forecasts into IT resource requirements.

vi. Map requirements onto existing utilizations.

vii. Predict when the IT will be out of capacity.

viii. Update forecasts and utilizations.

3.2 Why is it important?

Effective capacity planning allows IT operations to:

i. Optimize use of existing infrastructure.

ii. Lower capital expenses by accurately predicting future capacity resource requirements.
iii. Properly and efficiently plan for new infrastructure purchases.
iv. Improve service quality.
v. Eliminate redundant work.
vi. Ensure consistent reporting of performance and capacity.
vii. Provision capacity efficiently and in a timely manner.
viii. Uncover bottlenecks before business services are adversely affected.

### 3.3 How Capacity planning can help achieve EA goals

In chapter 2 we saw the outcome of a good EA practice. We will now discuss how capacity planning helps EA in achieving all those outcomes.

**Goal 1:** To provide with a more efficient business operation.

By efficiently utilizing the existing infrastructure resources, capacity planning helps lower business operational costs and improves business productivity.

**Goal 2:** To provide with a more efficient IT operation.

By automating the process of capacity planning, the software support and maintenance costs are lowered. Also, migration of applications becomes easy leading to increased portability of applications. Since the process is improved, it leads to easier system and network management.
**Goal 3:** To ensure better return on existing investment, reduced risk for future investment.

The capacity planning process tries to efficiently utilize the existing resources/infrastructure to give maximum productivity. It avoids investment in new infrastructure when the current infrastructure is sufficient. Thus, it provides maximum return on existing investment.

**Goal 4:** To help in faster, simpler, and cheaper procurement

Depending upon the forecasted workload, the capacity planning process can predict when and how much of capacity would be required in the future. Thus, it helps in making buying decisions as well.
Chapter 4: Related Work

This chapter talks about the related work and gives a brief description about various related topics.

**Topic Areas:** Application Resource Requirement Prediction, Capacity Planning for virtual environments.

*Predicting Application Resource Requirements in Virtual Environments*

Given resource utilization traces of an application running natively, the paper talks about estimating what its resource requirements would be if the application were transitioned to a virtual environment on a given hardware platform. They consider utilization metrics such as CPU (User Space %, Kernel %, IO Wait %) Network (Rx packets/sec, Tx packets/sec, Rx bytes/sec, TX bytes/sec) and Disk (Read req/sec, Write req/sec, Read blocks/sec, Write blocks/sec). A time series of measurements for each of these metrics is gathered and then analyzed.
'Profiling Applications for Virtual Machine Placement in Clouds'

This paper presents a novel application profiling technique using the canonical correlation analysis (CCA) method, which identifies the relationship between application performance and resource usage. They devise a performance prediction model based on application profiles generated using CCA. This talks more about co-locating applications on a particular data center in order to maximize system performance and application performance. They gather metrics such as CPU (%) I/O Read (MB/s) I/O Write (MB/s) Memory (MB), Transaction rate (#/s) Transaction time (s) for each application.

'Black-Box Approach to Capacity Identification for Multi-Tier Applications Hosted on Virtualized Platforms'

In this paper, the authors propose and evaluate a black-box method for capacity prediction that first identifies workload patterns for a multi-tier Web application from access logs using unsupervised machine learning and then, based on those patterns, builds a model capable of predicting the application’s capacity for any specific workload pattern. They use historical access logs to identify URIs with similar resource utilization characteristics.
'Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters'

This paper describes an approach to workload classification and its application to the Google Cloud Backend, arguably the largest cloud backend on the planet. The methodology for workload classification consists of: (1) identifying the workload dimensions; (2) constructing task classes using an off-the-shelf algorithm such as k-means; (3) determining the break points for qualitative coordinates within the workload dimensions; and (4) merging adjacent task classes to reduce the number of workloads.

The authors use the foregoing, especially the notion of qualitative coordinates, to glean several insights about the Google Cloud Backend: (a) the duration of task executions is bimodal in that tasks either have a short duration or a long duration; (b) most tasks have short durations; and (c) most resources are consumed by a few tasks with long duration that have large demands for CPU and memory.

‘An Optimized Capacity Planning Approach for Virtual Infrastructure Exhibiting Stochastic Workload’

Traditionally, any capacity planning problem is modeled with deterministic workloads by considering the peak workload for resource allocation. In the context of businesses using cloud service, cloud provider could allocate resources for peak workload which could lead to under utilization of resource and charging users for unused yet provisioned resources. In this paper, the authors came up with a better capacity planning algorithm which could ensure that they plan for peak usage but do not provision for it.
CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems

In this paper the authors present CloudScale, a prediction-driven elastic resource scaling system for multi-tenant cloud computing. The goal of the research is to develop an automatic system that can meet the SLO requirements of the applications running inside the cloud with minimum resource and energy cost. A description of an application-agnostic, light-weight online resource demand predictor is made and it is shown that it can achieve good prediction accuracy for a range of real world applications.

Automated Simulation-Based Capacity Planning for Enterprise Data Fabrics

In this paper, the authors present a novel case study of a representative EDF, the GemFire Enterprise, presenting a simulation-based tool that is developed for automated performance prediction and capacity planning. The tool automates resource demand estimation, performance model generation, performance model analysis and results processing. Given a system configuration and a workload scenario, the tool generates a report showing the predicted system throughput, server and network utilization, and operation response times. Both, the modeling approach and the model extraction technique are evaluated to demonstrate their effectiveness and practical applicability.

Application-aware Cross-layer Virtual Machine Resource Management

This paper proposes cross-layer optimization in VM resource management which allows certain awareness and cooperation between host and guest in order to improve application
performance and meet its QoS target. Specifically, this paper studies two aspects of such cross-layer optimization. First, guest-to-host optimization exploits guest-layer application knowledge to capture dynamic workload characteristics and improve the modeling of VM resource usage. Second, host-to-guest optimization enables the host-layer scheduler to feedback resource allocation decision and adapt guest-layer application configuration. These two aspects of cross-layer optimization are integrated into a fuzzy-modeling-based resource management system which uses fuzzy logic to model VM resource demand online and allocate resource dynamically according to application QoS requirement.

In these papers the authors deploy applications in physical as well as virtual environments and they gather usage statistics for both these environments using some monitoring tools. Depending upon the collected metrics, they find out a relationship between capacity required on the physical and virtual environments and build a model for capacity prediction.

However, our approach will be somewhat different. We would like to predict the resource usage before the application is actually deployed onto the virtual environment. Hence we only consider the usage statistics in the physical environment.
Chapter 5: Case Study - Nationwide

Our research embraces the innovative concept of using the Enterprise Architecture framework for capacity planning. In this chapter we will discuss the case study and the findings from it. We will look at the Enterprise Architecture and Capacity Planning AS-IS Scenarios. We will discuss the problems in the AS-IS scenario.

5.1 Enterprise Architecture AS-IS Scenario

The current Enterprise Architecture Repository has definitions such as Business Service, Application Component, Infrastructure Component, Technology Component etc. as shown in the Content Metamodel. We will be discussing about the Application and Infrastructure Components which play a major role in the process of Capacity Planning.

Some of the parameters each Application Component has are discussed below:

i. Introduction: As shown in figure 4 the Introduction tab contains fields such as 'Description' and 'Type' of Application.

ii. Decomposition: As shown in figure 5 the Decomposition tab contains fields such as 'Parent Application' and 'Child Applications'.
iii. Operational: As shown in figure 6 the Operational tab contains fields such as 'Organization Unit owner' of that Application.

iv. Service: As shown in figure 7 the Service tab contains fields such as 'Provides IS Services' and 'Acts as a Mediator'.

v. Technology: As shown in figure 8 the Technology tab contains fields such as 'Implemented on Technology Components' and 'Supported by Technology Components'.

Figure 4: Application Component - Introduction tab
Figure 5: Application Component - Decomposition tab

Figure 6: Application Component - Operational tab
Figure 7: Application Component - Service tab

Figure 8: Application Component - Technology tab
Some of the parameters each Infrastructure Component has are discussed below:

i. **Introduction:** As shown in figure 9 the Introduction tab contains fields such as 'Description', 'Server Type', 'Server Environment', 'Server Hardware', 'Operating System' of an Infrastructure Component.

![Diagram of Infrastructure Component - Introduction tab](image.png)

**Figure 9: Infrastructure Component - Introduction tab**

ii. **Operational:** As shown in figure 10 the Operational tab contains fields such as 'Located at' which gives the location of the Infrastructure Component.
iii. Application: As shown in figure 11 the Application tab contains fields such as 'Deploys Application Components' which gives a list of Applications deployed on the Infrastructure Component.
The operational data related to applications is not present in the Enterprise Architecture Repository. Such data is stored in other repositories such as Athena (Configuration Management Database), Sitescope (monitoring tool which monitors servers and collects usage data).

Following is the operational data for Application Components which is missing from EA.

i. Number of transactions per sec
ii. Maximum throughput
iii. CPU required per transaction
iv. Memory required per transaction
v. Average Transaction response time
vi. Operating System supported
vii. Web server supported

Following is the data for Infrastructure Components which is missing from EA.

i. Model
ii. Web server
iii. CPU cores
iv. Available CPU
v. Available Memory

The main problem is that the relationships between an Application Component and an Infrastructure Component are built in the real world and then they are mapped to EA i.e. EA has the static representation of the relationships.

Our idea is to reverse this process. Our research shows that we can use EA as a powerful tool to predict relationships between Application Component and Infrastructure Component. Using all the missing parameters mentioned above, EA will be able to predict the relationships and then these relationships can be built in the real world.
5.3 Capacity Planning AS-IS Scenario

**Capacity Planning for physical servers:**

At Nationwide, while deploying a new application onto the physical servers for the first time, an architect and/or a solution engineer typically are assigned and they use their experience and familiarity with applications similar to the new one to create a solution for the application. They make the initial server sizing decisions on their own in conjunction with the application owner and/or vendor if it is a 3rd party purchased application. Usually they start by supplying a small virtual test/dev environment for the application. Once a test environment is established the capacity planning team may get involved at the request of the architect or solution engineer to monitor a small load test. The results of this test along with a forecast of production load from the application area are used to determine the size required for the production/physical server.

**Capacity Planning for virtual servers:**

While migrating an application from the physical environment to the virtual environment, the capacity planning team identifies the infrastructure (physical servers) currently being used by that application. They gather data such as CPU, Memory capacity of those physical servers. Using the SPEC industry benchmark, they find out the Spec Rate of these physical servers and the equivalent Spec Rate of virtual servers. The application is then deployed on the virtual servers matching this Spec Rate.
5.4 Problem of Over-provisioning

In the approach discussed above, there is a problem of initial over sizing of the physical servers. Since the same requirements are translated into the virtual environment, there are problems of over provisioning which leads to wastage of useful resources. Our research aims at detecting the flaws in initial sizing and addresses them prior to shifting to production. We plan to implement a method which will help determine the sizing using application requirements, so as to eliminate the problems of over provisioning as well as under provisioning.

Figure 12 and Figure 13 show the CPU and Memory utilization of a nzvmas735 which is a virtual machine. Only 4 applications are deployed on this virtual machine. The CPU as well as memory utilization is very low.

![CPU Utilization Graph](image)

**Figure 12: Over provisioning - Low CPU Utilization**

32
As shown in Figure 12, the maximum CPU utilization is 5%.

The total memory allocated to this virtual machine is 6.9GB out of which only 2.2GB is used by the 4 applications as shown in Figure 13. Thus, 70% of the memory remains unused. This is a classic example of over-provisioning of valuable resources.

Figure 13: Over provisioning - Low Memory Utilization
Chapter 6: Research and Contributions

This chapter describes the research done and the contributions of this work towards the capacity planning problem. It discusses the approach, algorithm, challenges and results.

6.1 Problem Analysis

The problems seen so far are as follows:

i. Enterprise Architecture does not have application operational data e.g. usage metrics like CPU, memory etc.

ii. Application related operational data and infrastructure data are stored in different repositories such as APRM, Athena (CMDB) and Server Information Database.

iii. Capacity Planning for physical servers is done manually by Solution Engineers, Application owner based on previous experience.

iv. The same capacity is translated to the virtual environment which leads to wastage of valuable resources.

Our research tries to address all these problems.
6.2 Goals and Methodology

The final outcome of our research is a capacity plan. As shown in Figure 14, a user (Application owner at Nationwide) would submit an application deployment request. Our system would predict how much capacity the application would require in a virtual environment based on the usage statistics of that application on a physical environment. We will also predict the virtual machine in the North Data Center where the application can be deployed.

Figure 14: Goals

We can achieve this using the following methods:

1. Application Profiling
2. Capacity Prediction
3. Deployment Prediction
As shown in figure 15, our system gathers data from the following repositories:

**Athena:** Front end web interface into the CMDB (Configuration Management Database) and integrates data from various other sources. It provides configuration management information.

![Figure 15: System Architecture](image)

Figure 15: System Architecture
APRM: Source for application information.

Server Information Database (SID): A lotus domino application that tracks enterprise servers; from acquisition, changes, to surplusing of the assets. It serves as a data repository for server information.

Sitescope: Monitors and collects performance information for servers and applications.

Application logs: The logs contain usage data for each transaction.

The relationships (application to application dependencies) are pulled from the Enterprise Architecture Content Metamodel.

The data collected from the various sources will be stored in the Architecture Repository.

The EA Content Metamodel will help understand the relationships between all the components. Our framework will help the Business & IT executives in making decisions regarding modifications in business, investment in infrastructure etc. It will also help the capacity planning team, solution architects in planning & managing the infrastructure.

We have divided the capacity planning framework into three modules:

1. Application Profiling: Based on the CPU usage per transaction and memory usage per transaction, this module classifies the application as ‘CPU intensive’ or ‘Memory intensive’. This helps in preventing co-location of two CPU intensive or two Memory intensive applications on the same Virtual Machine.
2. **Capacity prediction:** For each application, using the maximum transaction rate and CPU, memory required for each transaction, the total CPU and Memory requirements are calculated.

3. **Deployment prediction:** This module predicts the Virtual Machine that an application can be deployed on. The attributes that will be taken into consideration are CPU, memory requirements of each application and the CPU, memory availability of the Virtual Machine. This module will make sure that a free pool of resources is reserved on each server to accommodate the dynamically changing application workload due to changing business needs. It will also allocate backup resources to each application based on its maximum throughput.

### 6.4 Parameters required for effective Capacity Planning

Figure 16 and 17 show the parameters that we have identified as the parameters required for effective capacity planning.

For each Application Component we should have the following parameters.
For each Infrastructure Component we should have the following parameters.
Using the parameters identified above, we have developed an algorithm which can accurately predict the capacity requirements of an application and also predict the virtual machine on which a particular application can be deployed. The algorithm steps are as follows:

1. Sort the applications based on the criticality rating and the number of users. Using this sorted list, find out the capacity requirements of each application and then deploy the application.
2. For application profiling: Based on the CPU and memory usage per transaction, classify the as CPU or Memory intensive or None.

3. For calculating capacity requirements of an application.

   Calculate CPU requirement & Memory requirement as follows:

   \[
   \text{CPU\_Req} = \text{Max\_Tx\_per\_sec} \times \text{CPU\_per\_Tx} \\
   \text{Mem\_Req} = \text{Max\_Tx\_per\_sec} \times \text{Mem\_per\_Tx}
   \]

4. Using the Data Center name, get a list of VMs where VM model, webserver, OS are same as that of the application to be deployed.

5. For deploying the application

   a) If the VM does not have any other application of the same category then

      - Check if the Virtual Machine has enough CPU and Memory (Consider VM reserved CPU, Memory and application backup CPU, memory).
      - Do not deploy more than one C1 criticality application on the same virtual machine.

   b) Deploy the application on several Virtual Machines depending upon its HA factor.

   c) After the application is deployed, check whether all its dependencies have already been deployed. If not, follow steps 2 to 5 for each of the application that it depends on.
Figure 18: Capacity Planning Algorithm
6.6 Challenges

For validating our algorithm, we needed data for some applications and infrastructure. We faced several challenges while finding the right data. Some of the challenges were as follows:

i. Data was to be pulled from several repositories. Also, each repository had different parameter values for the same component.

ii. Data was incomplete / inaccurate.

iii. Data was missing. For certain parameters, the measurement metrics were not present while in some cases operational data for a certain period was not gathered.

iv. Some operational data such as usage statistics per transaction are not monitored for the applications. For the purpose of validation, we had to get these statistics from application logs.

v. We found several discrepancies in Application names. e.g. an application 'Auto Acquisition' in Sitescope is known as 'Internet Auto Express' in Athena.

vi. Athena also had some application to infrastructure mappings which were incorrect (non-existent in real world).

6.7 Data Collection

We collected data for 5 applications and 3 virtual machines. Using the application usage statistics such as number of transactions, CPU, memory per transaction collected from the
physical environment, we predict the application capacity requirements on the virtual environment.

Table 1 shows the data that we collected for two of the applications.

Table 1: Application Components

<table>
<thead>
<tr>
<th>Application Id</th>
<th>1444</th>
<th>5363</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Internet Auto Express</td>
<td>Internet Auto Express Responsive</td>
</tr>
<tr>
<td>Description</td>
<td>Internet facing application for customers to quote an auto policy and bind it online in the following states: AZ, AR, CA, CT, DE, KY, MD, MI, MS, NC, NH, NY, NV, OK, PA, RI, SC, TX, VA, VT, and WV.</td>
<td>Auto Express Responsive is an internet facing application for customers to quote an auto policy and bind it online in AL, GE, IL, IN, KS, MO, TN, UT and WI. This application is a replacement for Internet Auto Express and focuses on responsive design to make sure our customers have seamless experience regardless of what device they use to get a quote.</td>
</tr>
<tr>
<td>LOB</td>
<td>NI</td>
<td>NI</td>
</tr>
<tr>
<td>DC Name</td>
<td>EDC-North</td>
<td>EDC-North</td>
</tr>
<tr>
<td>Criticality</td>
<td>C2</td>
<td>C2</td>
</tr>
<tr>
<td>No. of users</td>
<td>159790841</td>
<td>159790841</td>
</tr>
<tr>
<td>Max throughput</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Max Tx per sec</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>CPU per Tx (%)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Mem per Tx (MB)</td>
<td>333</td>
<td>500</td>
</tr>
<tr>
<td>OS supported</td>
<td>Linux</td>
<td>Linux</td>
</tr>
<tr>
<td>Model supported</td>
<td>s390x</td>
<td>s390x</td>
</tr>
<tr>
<td>Web server supported</td>
<td>IBM WebSphere Application Server</td>
<td>IBM WebSphere Application Server</td>
</tr>
<tr>
<td>HA Factor</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Table 2 shows the virtual machines (infrastructure components) that we considered for deployment.

Reserved CPU and Reserved Memory are the resources that we have decided to reserve for each machine. It is 20% of the total available resources.

The last three rows show the current deployment scenario - i.e. current CPU and memory utilization.

<table>
<thead>
<tr>
<th>Name</th>
<th>nzvmas735</th>
<th>nzvmas742</th>
<th>nzvmas743</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC Name</td>
<td>EDC-North</td>
<td>EDC-North</td>
<td>EDC-North</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>6900</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>CPU cores</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>OS</td>
<td>Linux</td>
<td>Linux</td>
<td>Linux</td>
</tr>
<tr>
<td>Model</td>
<td>s390x</td>
<td>s390x</td>
<td>s390x</td>
</tr>
<tr>
<td>Web server</td>
<td>IBM WebSphere Application Server</td>
<td>IBM WebSphere Application Server</td>
<td>IBM WebSphere Application Server</td>
</tr>
<tr>
<td>Reserved CPU (%)</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Reserved Mem (MB)</td>
<td>1000</td>
<td>750</td>
<td>750</td>
</tr>
<tr>
<td>Applications List</td>
<td>AgentOpp, Internet Auto Express, Internet Auto Express Responsive, eBill</td>
<td>AWP, AgentOpp, Internet Auto Express, Internet Auto Express Responsive, eBill</td>
<td>AWP, AgentOpp, Internet Auto Express, Internet Auto Express Responsive, eBill</td>
</tr>
<tr>
<td>CPU Utilization (%)</td>
<td>7</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Memory Utilization (MB)</td>
<td>2100</td>
<td>2200</td>
<td>2100</td>
</tr>
</tbody>
</table>
6.8 Results and Validation

We implemented our algorithm as a standalone Java application and used MySQL as the database for storing the application and infrastructure data.

Using the data collected, we executed our algorithm and got the following results:

i. The 3 virtual machines that we considered originally had 5 applications deployed across them. We considered these 5 applications and 4 dummy applications with similar profiles.

ii. Using our algorithm, we predicted that 8 applications can be deployed on those 3 virtual machines with SLAs satisfied.

Figure 19 shows the memory utilization on those 3 virtual machines before and after predictions using our algorithm. From the graphs we see that the memory utilization has increased by about 40-50%.

Figure 20 shows the CPU utilization on those 3 virtual machines before and after predictions using our algorithm. Total CPU is 300% for nzvmas735 since there are 3 CPU cores on it. For nzvmas742 and nzvm743 the total CPU is 200% since there are 2 CPU cored on each. From the graphs we see that the CPU utilization has increased by about 8-12%.
Figure 19: Results - Memory Utilization

(a) 30-44% memory utilization
56-70% unused

(b) 82-84% memory utilization (After reserving backup for each application)
16-18% unused
Figure 20: Results - CPU Utilization

(a)  
2-3% CPU utilization  
97-98% unused

(b)  
10-15% CPU utilization  
85-90% unused
Summary of results achieved

This section summarizes the results that we have achieved. We have identified the parameters which are required for effective capacity planning. Using the related work and the validation of our algorithm, we can conclude that these parameters are correct. Our validation also shows that our algorithm can produce better results than the current scenario, i.e. if our predictions are followed; it will result in better CPU and memory utilization along with satisfaction of SLAs for the applications. We have also analyzed what parameters and relationships EA has and which ones are missing. We can add these missing parameters to EA so that EA can be used for effective capacity planning. Thus, we have proved that EA can be used as a powerful tool for capacity planning.
Chapter 7: Business Value

This chapter discusses the effectiveness of this work and how this work benefits the Java hosting team and capacity planning team at Nationwide. It also describes the business value of our research at various levels of the Enterprise Architecture.

7.1 Service Blueprint

First we discuss how Enterprise Architecture plays a major role in the process of Capacity Planning. This is shown in the Figure 21 which is our Service Blueprint.

- **Process to be blueprinted**: Capacity planning.
- **Customer segment**: Application Owners at Nationwide.
- **Front office employees**: Capacity Planning team.
- **Back office employees**: Project Manager, Developers.
- **Supporting services**: Infrastructure team, CMDB Athena, SID, APRM

As shown in Figure 21, the Application owners form the customer segment. They provide the capacity planning team with the Architectural Requirements and other deployment
requirements of the application to be deployed. The capacity planning team understands the requirements, gathers all the AS-IS information and provides it to the back office employees. The line of interaction lies between the application owners and the capacity planning team. The line of visibility lies between the capacity planning team and the back office employees. The project manager then modularizes the development process and divides the workload among the developers. The application profiling is done followed
by the capacity and deployment predictions. This plan is then submitted to the capacity planning team which provides the application owners with the deployed application which satisfies SLAs. Below the line of internal interaction is the Enterprise Architecture view. It consists of all the data from the different repositories discussed earlier as well as the relationships from the current Enterprise Architecture repository. These are the major supporting services for predicting the application capacity requirements and deployment.

7.2 Benefits to the Java Hosting team

The Java Hosting Environment is a Nationwide collaboration with IBM to redesign and implement how Nationwide hosts Java applications. It will establish a world-class hybrid private cloud that will enable Nationwide to run and manage Java workloads on the most efficient, appropriate cost effective architecture, based on application characteristics. The aim of the Java Hosting team is to enable Java applications to run in elastic virtual environments spanning across VMWare and zLinux. Their goal is to have a reduced application impact resulting from infrastructure maintenance; a move towards continuous availability. They need a simplified creation of Java testing resources, thus reducing provisioning time. Nationwide has about 2000 applications out of which currently 7 applications have been moved to the Java Hosting Environment. The process of migrating the rest of the applications is ongoing. Our research algorithm will help the Java Hosting team in predicting the capacity (CPU, memory) requirement for every application and it will also help in determining the virtual machines on which the
application and its dependencies can be deployed. Using this initial prediction, the applications can be migrated and later on, monitoring tools can be used to determine if there is a need for dynamic re-allocation of resources.

7.3 Business value at each level in the Enterprise Architecture

Figure 22 shows the business value of our research at various levels in the Enterprise Architecture.

At the lowest level i.e. infrastructure, the capacity planning algorithm will help in easier system and network management. At the application level it helps in accurate resource prediction and satisfaction of customer SLAs. At the Business Process level it helps in reducing the complexity in business and IT by automating the capacity prediction process. Finally at the Business Service level it helps in lowering the infrastructure and operational costs and gives better Return on Investment from the existing infrastructure.
Figure 22: Business Value

- **Business Services**: Lower business operation costs
- **Business Processes**: Maximum ROI in existing business and IT infrastructure
- **Applications**: Reduced complexity in the business and IT
- **Infrastructure**: Satisfaction of customer SLAs
- **Infrastructure**: Easier system and network management
Chapter 8: Conclusion and Future Work

Thus, we have successfully identified the parameters required for effective capacity planning and also identified the ones missing from Enterprise Architecture. Our research embraces an innovative concept of how Enterprise Architecture can be used for solving the problem of capacity planning. We have also implemented and validated our algorithm for capacity planning. Our algorithm improves the process of capacity planning by automating it, thereby reducing manual efforts. It helps in eliminating the problem of over provisioning by maximizing the utilization of available resources. The result of this will be a decrease in operational costs due to higher utilization of existing infrastructure. This also helps the end users i.e. Application Owners at Nationwide since the application deployment process becomes much easier.

This problem needs enough work to be done in the future. Some of which can possibly be the following steps:

- Use of data mining tools for application profiling.
- Identifying the effect of other components such as Business Process, IS Service etc. from the Enterprise Architecture on the need for Infrastructure resources.
- Considering the effect of changing business needs while predicting capacity.
References


11. http://www.opengroup.org/togaf/


15. Dataset provided by Nationwide capacity planning team and application teams.
APPENDIX A: Details of the algorithm
This section describes in detail, the algorithm shown in Figure 18.

1. The Application sorting module sorts the applications based on the criticality rating and number of users. Each application has a criticality rating between C1 to C5. C1 is the most critical rating. When we have a list of applications to be deployed, we give preference and predict capacity and deployment for the highly critical applications and applications with large number of users first.

2. The Application Profiler classifies the applications as CPU/Memory intensive or none. This is used for co-location of applications on a virtual machine. We do not place two CPU intensive or two memory intensive applications on the same virtual machine.

3. Using the maximum transaction rate and resource usage statistics per transaction, we find out the total CPU and memory requirements for each application.

4. For each application we find out a list of virtual machine where the VM model, OS and webserver match with the application model, OS and webserver requirements.

5. From the list of VMs generated in step 4, we select each VM and check whether the Vm can satisfy the application’s resource requirements. For each VM, we keep 20% of resources reserved. We also have backup resources for each application. This is decided based on the maximum throughput i.e. the maximum number of transactions the application is capable of handling.
6. If the application is C1 criticality application, we see to it that we select a VM where there is no other C1 criticality application already deployed.

7. We also see to it that two applications of the same category (CPU intensive or memory intensive) are not placed on the same virtual machine.

8. If all these requirements are satisfied, then we predict that the application can be deployed on that particular VM.

9. We also consider HA i.e. High Availability factor. We predict deployment of the application on number of VMs same as that of the HA factor.

10. Lastly, we check if all the application dependencies are already addressed. If the applications in the dependency list are not deployed, we repeat steps 2 to 9 for each application in the dependency list.
APPENDIX B: Java Implementation
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

public class ApplicationComponent implements Comparable<ApplicationComponent> {
    private int id;
    private String name;
    private String LOB;
    private String DC_Name;

    private String criticality;
    private int no_of_users;
    private int max_Throughput;
    private int max_Tx_per_sec;
    private String OS_supported;
    private double mem_per_Tx;
    private int CPU_per_Tx;
    private String model_supported;
    private int CPU_cores_needed;
    private String webserver_supported;
    private String category;
    private double mem_req;
    private int CPU_req;
    private int backup_CPU;
    private double backup_mem;
    private int HAFactor;
    private boolean deployed;
    private ArrayList<ApplicationComponent> dependencies;

    public int getId() {
        return id;
    }
    public boolean isDeployed() {
        return deployed;
    }
    public void setId(int id) {
        this.id = id;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setLOB(String LOB) {
        LOB = LOB;
    }
    public void setDC_Name(String dC_Name) {
        DC_Name = dC_Name;
    }
    public void setCriticality(String criticality) {
this.criticality = criticality;
}
public void setNo_of_users(int no_of_users) {
    this.no_of_users = no_of_users;
}
public void setMax_Throughtput(int max_Throughtput) {
    this.max_Throughtput = max_Throughtput;
}
public void setMax_Tx_per_sec(int max_Tx_per_sec) {
    this.max_Tx_per_sec = max_Tx_per_sec;
}
public void setOS_supported(String oS_supported) {
    OS_supported = oS_supported;
}
public void setMem_per_Tx(double mem_per_Tx) {
    this.mem_per_Tx = mem_per_Tx;
}
public void setCPU_per_Tx(int cPU_per_Tx) {
    CPU_per_Tx = cPU_per_Tx;
}
public void setModel_supported(String model_supported) {
    this.model_supported = model_supported;
}
public void setCPU_cores_needed(int cPU_cores_needed) {
    CPU_cores_needed = cPU_cores_needed;
}
public void setWebserver_supported(String webserver_supported) {
    this.webserver_supported = webserver_supported;
}
public void setBackup_CPU(int backup_CPU) {
    this.backup_CPU = backup_CPU;
}
public void setBackup_mem(double backup_mem) {
    this.backup_mem = backup_mem;
}
public void setDeployed(boolean deployed) {
    this.deployed = deployed;
}
public void setDependencies(ArrayList<ApplicationComponent> dependencies) {
    this.dependencies = dependencies;
}

public int getBackup_CPU() {
    return backup_CPU;
}
public double getBackup_mem() {
    return backup_mem;
}
public String getName() {
    return name;
}
public String getLOB() {
    return LOB;
}
public String getDC_Name() {
    return DC_Name;
}

public String getCriticality() {
    return criticality;
}

public int getNo_of_users() {
    return no_of_users;
}

public int getMax_Throughtput() {
    return max_Throughtput;
}

public int getMax_Tx_per_sec() {
    return max_Tx_per_sec;
}

public String getOS_supported() {
    return OS_supported;
}

public double getMem_per_Tx() {
    return mem_per_Tx;
}

public int getCPU_per_Tx() {
    return CPU_per_Tx;
}

public String getModel_supported() {
    return model_supported;
}

public int getCPU_cores_needed() {
    return CPU_Cores_needed;
}

public String getWebserver_supported() {
    return webserver_supported;
}

public double getMem_req() {
    return mem_req;
}

public void setMem_req(double mem_req) {
    this.mem_req = mem_req;
}

public int getCPU_req() {
    return CPU_req;
}

public void setCPU_req(int CPU_req) {
    CPU_req = CPU_req;
}

public String getCategory() {
    return category;
}

public void setCategory(String category) {
    this.category = category;
}

public int getHAFactor() {
public void setHAFactor(int factor) {
    this.HAFactor = factor;
}

dependencies = new ArrayList<ApplicationComponent>();

dependencies.add(new ApplicationComponent(criticality, no_of_users, max_Throughtput, max_Tx_per_sec, mem_per_Tx, CPU_per_Tx, model_supported, OS_supported, webserver_supported, backup_CPU, backup_mem, HAFactor, deployed));

public int compareTo(ApplicationComponent other) {
    int result = this.criticality.compareTo(other.criticality);
    if (result == 0) {
        Integer this_no = new Integer(this.no_of_users);
        Integer a_no = new Integer(other.no_of_users);
        result = this_no.compareTo(a_no);
    }
    return result;
}

public void initApp(ResultSet res) {
    try {
        this.setId(res.getInt("app_id"));
        this.setName(res.getString("appname"));
        this.setLOB(res.getString("LOB"));
        this.setDC_Name(res.getString("DC_name"));
        this.setCriticality(res.getString("Criticality"));
        this.setNo_of_users(res.getInt("no_of_users"));
        this.setMax_Throughtput(res.getInt("max_Throughtput"));
        this.setMax_Tx_per_sec(res.getInt("max_Tx_per_sec"));
        this.setMem_per_Tx(res.getDouble("mem_per_Tx"));
        this.setCPU_per_Tx(res.getInt("CPU_per_Tx"));
        this.setModel_supported(res.getString("model_supported"));
        this.setOS_supported(res.getString("OS_supported"));
        this.setWebserver_supported(res.getString("webserver_supported"));
        this.setBackup_CPU(res.getInt("backup_CPU"));
        this.setBackup_mem(res.getDouble("backup_mem"));
        this.setHAFactor(res.getInt("HAFactor"));
        this.setDeployed(res.getBoolean("deployed"));
    } catch (SQLException e) {
        // TODO Auto-generated catch block
    }

    dependencies.add(new ApplicationComponent(criticality, no_of_users, max_Throughtput, max_Tx_per_sec, mem_per_Tx, CPU_per_Tx, model_supported, OS_supported, webserver_supported, backup_CPU, backup_mem, HAFactor, deployed));
}

return HAFactor;
}
VirtualMachine.java

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

public class VirtualMachine extends InfrastructureComponent {

    private String OS;
    private double available_memory;
    private int available_CPU;
    private int CPU_cores;
    private String webserver;
    private String model;
    private ArrayList<ApplicationComponent> appsList;
    private int reserved_CPU;
    private double reserved_memory;

    public void setReserved_CPU(int reserved_CPU) {
        this.reserved_CPU = reserved_CPU;
    }
    public void setReserved_memory(double reserved_memory) {
        this.reserved_memory = reserved_memory;
    }

    public String getOS() {
        return OS;
    }
    public void setOS(String oS) {
        OS = oS;
    }

    public double getAvailable_memory() {
        return available_memory;
    }
    public void setAvailable_memory(double available_memory) {
        this.available_memory = available_memory;
    }

    public int getAvailable_CPU() {
        return available_CPU;
    }
    public void setAvailable_CPU(int available_CPU) {
        this.available_CPU = available_CPU;
    }

    public int getCPU_cores() {
        return CPU_cores;
    }
}
public void setCPU_cores(int CPU_cores) {
    CPU_cores = CPU_cores;
}

public String getWebserver() {
    return webserver;
}

public void setWebserver(String webserver) {
    this.webserver = webserver;
}

public String getModel() {
    return model;
}

public void setModel(String model) {
    this.model = model;
}

public ArrayList<ApplicationComponent> getAppsList() {
    return appsList;
}

public void setAppsList(ArrayList<ApplicationComponent> appsList) {
    this.appsList = appsList;
}

public int getReserved_CPU() {
    return reserved_CPU;
}

public double getReserved_memory() {
    return reserved_memory;
}

public void initVM(ResultSet res) {

    try {
        this.setId(res.getInt("VM_id"));
        this.setName(res.getString("VM_name"));
        this.setDC_Name(res.getString("DC_Name"));
        this.setOS(res.getString("OS"));
        this.setAvailable_CPU(res.getInt("available_CPU"));
        this.setCPU_cores(res.getInt("CPU_cores"));
        this.setWebserver(res.getString("Webserver"));
        this.setModel(res.getString("model"));
        this.setReserved_memory(res.getDouble("reserved_memory"));
        this.setReserved_CPU(res.getInt("reserved_CPU"));
    }
    catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```
import java.util.ArrayList;

public class InfrastructureComponent {
    private int id;
    private String name;
    private String LOB;
    private String DC_Name;
    private String type;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getLOB() {
        return LOB;
    }
    public void setLOB(String lOB) {
        LOB = lOB;
    }
    public String getDC_Name() {
        return DC_Name;
    }
    public void setDC_Name(String dC_Name) {
        DC_Name = dC_Name;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
}
```
PhysicalServer.java

```java
import java.util.ArrayList;

public class PhysicalServer extends InfrastructureComponent {
    private int noOfVMs;
    private int available_CPU;
    private ArrayList<VirtualMachine> listOfVMs;

    public ArrayList<VirtualMachine> getListOfVMs() {
        return listOfVMs;
    }
    public void setListOfVMs(ArrayList<VirtualMachine> listOfVMs) {
        this.listOfVMs = listOfVMs;
    }
    public int getNoOfVMs() {
        return noOfVMs;
    }
    public void setNoOfVMs(int noOfVMs) {
        this.noOfVMs = noOfVMs;
    }
    public int getAvailable_CPU() {
        return available_CPU;
    }
    public void setAvailable_CPU(int available_CPU) {
        this.available_CPU = available_CPU;
    }
}
```

Connect.java

```java
import java.sql.*;

public class Connect{
```

69
public Connection getConnection() {
    try {
        Class.forName(driver).newInstance();
        conn = DriverManager.getConnection(url+dbName,userName,password);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return conn;
}

Client.java

import java.lang.reflect.Array;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Collections;
import java.sql.PreparedStatement;
public class Client {

    public static void main(String[] args) {
        Client c = new Client();
        String DC_Name = "";
        ArrayList<ApplicationComponent> apps = new ArrayList<ApplicationComponent>();
        // select * from ApplicationComponent and add in apps.
```java
Statement st;

try {
    Connect obj = new Connect();
    Connection conn = obj.getConnection();
    st = conn.createStatement();
    ResultSet res = st.executeQuery("SELECT * FROM applicationcomponent");

    while (res.next()) {
        ApplicationComponent app = new ApplicationComponent();
        app.initApp(res);
        apps.add(app);
    }
} catch (SQLException e) {
    e.printStackTrace();
}

apps = c.sortApps(apps);
for(ApplicationComponent a : apps) {
    c.classifyApp(a);
    c.calculateAppRequirements(a);
    ArrayList<VirtualMachine> VMlist = new ArrayList<VirtualMachine>();
    VMlist = c.getVMList(a);
    c.calculateDeployment(VMlist,a);
}

private void classifyApp(ApplicationComponent app) {
    if(app.getCPU_per_Tx()>30)
        app.setCategory("CPU");
    else if(app.getMem_per_Tx()>500)
        app.setCategory("Mem");
    else
        app.setCategory("none");
}

private void calculateAppRequirements(ApplicationComponent a) {
    int CPU_Req = a.getMax_Tx_per_sec() * a.getCPU_per_Tx();
    double Mem_Req = a.getMax_Tx_per_sec() * a.getMem_per_Tx();
    a.setCPU_req(CPU_Req);
    a.setMem_req(Mem_Req);
```
private void calculateDeployment(ArrayList<VirtualMachine> VMlist, ApplicationComponent a) {
    ArrayList<ApplicationComponent> deployedApps;
    boolean flag = false, C1flag = false;
    int HA = a.getHAFactor();
    if(a.getCriticality().equalsIgnoreCase("C1"))
        C1flag = true;
    for(VirtualMachine vm : VMlist)
    {
        if(HA!=0)
        {
            //check if same category app already present
            deployedApps = vm.getAppsList();
            if(deployedApps!=null){
                for(ApplicationComponent deployedApp :
                    deployedApps)
                    {
                        System.out.println("in deployed apps loop");
                        if(a.getCategory()!="None" &&
                        deployedApp.getCategory().equalsIgnoreCase(a.getCategory()))
                            flag = true;
                        //if app criticality == c1 check if there exists another C1 criticality app
                        else if (C1flag &&
                        deployedApp.getCriticality().equalsIgnoreCase("C1"))
                            flag = true;
                    }
                    if(flag)
                        continue;
                
                // else
                int CPU = vm.getAvailable_CPU() -
                vm.getReserved_CPU();
                double mem = vm.getAvailable_memory() -
                vm.getReserved_memory();
                ArrayList<ApplicationComponent> appsList;
                if(CPU >= (a.getCPU_req() + a.getBackup_CPU())
                && (mem >= (a.getMem_req() + a.getBackup_mem())))
                {
                    // Associate App with VM
                    System.out.println("Application " +
                    a.getName() + " can be deployed on VM " + vm.getName());
                }
            }
        }
    }
}
String sql = "update virtualmachine set available_CPU = ?, available_memory = ? where VM_id = ?";
PreparedStatement stmt;
Connect obj = new Connect();
Connection conn = obj.getConnection();

try {
    stmt = conn.prepareStatement(sql);
    stmt.setInt(1, vm.getAvailable_CPU()-a.getCPU_req()-a.getBackup_CPU());
    stmt.setDouble(2, vm.getAvailable_memory()-a.getMem_req()-a.getBackup_mem());
    stmt.setInt(3, vm.getId());
    stmt.executeUpdate();
}

} catch (SQLException e) {
    e.printStackTrace();
}

appsList = vm.getAppsList();
if(appsList!=null)
    appsList.add(a);
else {
    appsList = new ArrayList<ApplicationComponent>();
    appsList.add(a);
}
vm.setAppsList(appsList);

// Decrement HA Factor
HA-- ;
if(HA!=0)
    continue;

/*// Application Dependencies
for(ApplicationComponent dep :
    a.getDependencies()) {
    classifyApp(dep);
    calculateAppRequirements(dep);
    ArrayList<VirtualMachine> VMs = new ArrayList<VirtualMachine>;
    VMs = getVMList(a);
    calculateDeployment(VMs, dep);
} */
private ArrayList<VirtualMachine> getVMList(ApplicationComponent a) {
    // Using the Data Center name, get a list of VMs where VM model, 
    // webserver, OS, CPU cores are same as that of the application 
    // to be deployed.
    ArrayList<VirtualMachine> VMlist = new ArrayList<VirtualMachine>();
    
    String sql = "SELECT * FROM VirtualMachine WHERE DC_Name = ? & model = "" & "" & OS = "" & CPU cores >= "";
    PreparedStatement stmt;
    try {
        stmt = conn.prepareStatement(sql);
        stmt.setString(1, a.getDC_Name());
        stmt.setString(2, a.getWebserver_supported());
        stmt.setString(3, a.getOS_supported());
        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            VirtualMachine VM = new VirtualMachine();
            VM.initVM(rs);
            VMlist.add(VM);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return VMlist;
}
private ArrayList<ApplicationComponent> sortApps(ArrayList<ApplicationComponent> apps) {
    Collections.sort(apps);
    return apps;
}