Hosting Online Games on Wide-Area Computing Utilities Using Model-Based Resource Provisioning

Weiquan Yuan

School of Computer Science
McGill University
Montreal, Canada

February 2007

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

2007/02/02

Your file Votre référence
ISBN: 978-0-494-32807-1
Our file Notre référence
ISBN: 978-0-494-32807-1

Abstract

Constant improvements in computer communications, microprocessors and operating systems are finally making wide-area *public computing utilities* (PCUs) a reality. The core idea of a CU is to connect resources that are geographically distributed and potentially heterogeneous into a single networked system such that clients can obtain on-demand the necessary resources for their computing needs from this system. Several PCUs have been created in the recent past using the Grid technology for both research and commercial purposes. Because PCUs often bring together resources from geographically distributed locations, they are ideally suited for applications that have geographically distributed resource demands. An example application with geographically distributed demand is online gaming. A popular online game can be a meeting point for a large number of players from diverse network locations. Often the gameplay can generate network and process intensive transactions that should be carried out by the game servers in a timely fashion for positive player experience. Using PCUs and distributed game server architectures, it is possible to locate game servers closer to major concentrations of the player population, which can satisfy the quality of service requirements of vast majority of the players.

In this thesis, I present an approach for online game hosting on wide-area computing utilities. My approach is based on *virtual clusters* (VCs) that are equivalent to *virtual organizations* (VOs). A VC can be considered as a "slice" of the PCU resource pool that is allocated for a single game or class of games. Because I was only concerned about the resource allocation problem, I refer to the slice as VC instead of VO.

This study has two parts. In the first part, a performance model is developed for an online game that can be used for hosting purposes. Quake, as a sequential (from Quake I to Quake IV), publicly available, multiplayer game server became my study object because it is a commercial server application that has been used extensively and exhibits many of the required characteristics. The performance modeling methodology, however, is sufficiently generic and should be applicable to other online games. In the second part, a resource allocation algorithm is developed for multiple VCs creation. This algorithm is capable of creating multiple VCs with different demand profiles. The multiple VCs creation problem is equivalent to a facility location problem that is proven to be NP-hard, which is concerned with optimally placing a number of facilities (equivalent to serving resources) to cover demands at a predefined set of points. The assumption is that placing a facility at a given candidate location incurs a fixed cost and each facility has a limited capacity of covering the demands. An efficient heuristics is developed, which can be deployed at short time scales. To minimize the multiple VCs reconfigurations, redundancy is built into the multiple VCs model. Extensive simulation studies were used to evaluate the performance of the multiple VCs model. The experimental results prove that the developed algorithm for multiple VCs model can treat each VC in a fair and efficient way.

Résumé(Français)

L'volution constante dans le domaine des communications informatiques, des microprocesseurs et systmes dexploitation rendent finalement possible le dploiement des rseaux tendus des grilles informatiques (GI). L'ide centrale des GI est de permettre aux clients d'accder aux ressources informatiques d'innombrables units, quels que soient leurs endroits gographiques ou leurs systmes d'exploitation. De nombreux GI fin commerciale et de recherche fut rcemment cr, utilisant l'informatique en grille. La nature des GI se prte trs bien aux applications ayant besoin de ressources sur diffrente location gographique. Un exemple d'une telle application est les jeux en ligne. Certains jeux populaires deviennent un endroit de rencontre pour de nombreux joueurs provenant d'endroits divers. Le volume de trafic et la demande des ressources gnres par les jeux en ligne doivent tre traits rapidement par des serveurs afin de crer un rsultat satisfaisant pour le joueur. L'utilisation des GI coupl avec une architecture distribue des serveurs de jeux en ligne, permettent de localiser les serveurs le plus proche des concentrations de joueurs, permettant ainsi de rencontrer les exigences attendues des joueurs.

Je prsente avec cette thse une approche permettant d'hberger des jeux en ligne sur des GI en rseau tendu. Mon approche est constitue de grappe virtuelle (GV), quivalente aux organisations virtuelles. Une GV peut tre considr comme un 'morceau' dans la rserve de ressource qui est alloue pour un jeu ou une classe de jeu. Veuillez noter que je rfre au 'morceau' comme un GV au lieu d'un VO, tant donn que ma seule proccupation est d'adresser le problme de l'allocation des ressources.

Cette tude s'tend sur deux parties. La premire partie concerne le dveloppement d'un modle de performance pouvant tre utilise comme station d'hbergement pour les jeux en ligne. Aux fins de cette tude, le jeu en ligne Quake est tudi, car il rpond aux critres requis (application serveur commerciale, disponible au public en gnral, trs populaire et massivement multi joueur). Il est noter que la mthodologie utilise pour la modlisation de la performance est assez gnrique pour tre appliqu aux autres jeux en ligne. Dan la deuxime partie je dveloppe un nouvel algorithme d'allocation de ressources pour la cration des GV. Cet algorithme est capable de crer plusieurs GV, ayant chacun un diffrent profil de demande de ressources. tant donne la similitude entre la rsolution du problme de la cration des GV et du problme de la rsolution de la location de ressources, tous deux NPhard, de trs bonnes heuristiques furent dveloppes et dployes trs rapidement. Afin de minimiser la reconfiguration des GV inutilement, un modle de redondance fut aussi intgr dans le modle des GV. Des tudes extensives propos des simulations furent utilises dans l'valuation de la performance du modle des GV. L'exprimental les rsultats montrent que l'algorithme dvelopp pour le modle multiple de VCs peut traiter chaque VC d'une manire juste et efficace.

Acknowledgments

I would like to thank Prof. Muthucumaru Maheswaran for his continuous guidance and supervision on my thesis research. His advice and suggestions are invaluable for this thesis achievement.

Thanks should be given to everyone in the Advanced Network Research Lab for their valuable comments. Special thanks to Balasubramaneyam Maniymaran, Arindam Mitra and Shah Asaduzzaman, who provided lots of help on the testing and debugging of my program, general computer knowledge and thesis correcting.

I'd also like to express my thanks to Sheng Lu, who gave me much convenience and some advice during the process of thesis. Furthermore, I would like to thank Jacques Boucher and Steve McCutchen's unselfishness contribution on my thesis correction and content refinement.

And last, but not least, I convey my heartiest thanks to my parents Xiuling Cheng and Changping Yuan, my uncle ChangWu Yuan and my cousin Amy Yuan and her husband Rui Gong for their continuous moral support in my entire life and career.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet user base has increased to hundreds of million of people in recent years. AOL's Web caches service over 10 billion hits a day [1]. The number of concurrent sessions and hits per day to Internet sites results into an even higher number of I/O and network requests, which places enormous demands on underlying resources. Meanwhile, online game services such as PlayStation online, Xbox Live, GameSpy Arcade,and other independent PC-based services are becoming increasingly more popular than ever. Based on statistics on online games [2], popular online role-playing game titles have large numbers of subscribers, which has reached 120 million by July 2006. This existing situation makes deployment of game services that can provide responsive, robust, and continuous service to clients a computer systems problem of unprecedented scale.

Solving the game services deployment problem is challenging and resource intensive. The most common approach used by current service providers is to significantly overprovision the amount of resources needed to support such applications by tailoring the resources to maximum expected load. As this is not a good way to use Internet resources efficiently to meet dramatically growing demand, new system design techniques must be developed to manage game workloads.

One alternative approach is to design deployment infrastructures based on the "utility computing model," which automatically manages the resources used by the system based on the demand for the services. How to build such infrastructures is receiving more and more attention from academic and industrial communities which have been proposing innovative solutions and trying to implement them. The on-demand business model provided by IBM [3, 4] addresses similar problems with business applications, where the issue of infrastructure cost strongly motivates new on-demand business models for utility computing offerings. These models provide the flexibility to scale an application or service in response to user demand by rapidly adding or removing resources (e.g., servers, storage, databases, network bandwidth, etc.) from a pool that may be shared among multiple applications or customers. With this on-demand infrastructure, service providers could enjoy benefits by reducing initial investment, scaling rapidly according to demand and adding new services. For this technology, Hewlett-Packard is another active member in the community to line up. HP introduces a *Utility Data Center* (UDC) [5, 6] which provides a flexible, cost-effective solution by using advanced management software, allowing resources to be automatically reassigned in response to changing business and IT requirements. The UDC architecture is an ideal platform to support the efficient hosting of applications for Internet services. Its infrastructure provides a set of new management capabilities for requesting/releasing the system resources to dynamically provision the application demands.

There is a growing amount of research focusing on how to take advantage of this on-demand infrastructure to help traditional services benefit from the resource management. In [7] a new model-based approach to dynamically provisioning multiple resources that interact in complex ways for hosted Web services is demonstrated. Cherkasova et al. ([8]) show a workload-aware performance model of streaming media application in UDC infrastructure. Both experimental results positively show that on-demand consumption of multiple resources can be predicted to meet the service loads with the performance model, suggesting the *model-based resource provisioning* (MBRP) is flexible enough to adjust to resource constraints or surpluses exposed during resource assignment within the on-demand infrastructure. Because traditional services such as Web and online media services can benefit from it, online games might as well.

Online game services that enable multiple players to simultaneously interact in a "game world" face the same pain points related to the provision and demand problems traditional services have met recently. Due to the high risk of over-and under-allocation of resources and potentially poor resource utilization, gaming service providers have to install a dedicated infrastructure for each game title to meet the challenges brought by dynamics of the users. This situation may become worse as the number of commercially operated game titles has increased dramatically in recent years.

While not indicative of all online games, the "first-person shooter" (FPS) has clearly dominated much of the observed gaming industry. With a large list of popular games such as Quake, Doom, Half-Life, etc., these games are representative of what online games of the future are moving towards. In my work, Quake has been chosen as the game service to be investigated.

The topic of the thesis is how to apply the on-demand infrastructure to online game. Given the similarities between game and Web/media services, the *public computing utility* (PCU) model presented in [9] can be extended to solve the game hosting problem from an infrastructure perspective. PCU has unique ways of providing a cost-effective, guaranteed *quality of service* (QoS) and service specific resource management infrastructure, better suited for online game. For example, due to interactivity requirements, FPS games require that the server resources be placed geographically close to the players, which can be considered as a cost-driven and demand-driven resource allocation problem in PCU. Because the concentration of player populations can vary dramatically, QoS guaranteed by PCU is also critical to optimize server utilization while maximizing game playing satisfaction.

However, the existing models for Web or media services can not be applied directly, due to unique aspects of online gaming, such as player behavior, map complexity, game engine and game design. Therefore, a performance model was developed for the game Quake to address these and merged with an underlying PCU infrastructure. Another contribution of this work is a heuristic algorithm to create multiple virtual clusters to host multiple services at the same time. Additionally, while our architecture is geared initially toward FPS games, the performance model can also be applied to *massively multi-player online role playing games* (MMORPG) as well.

This thesis is organized as follows: Chapter 2 provides some related works on resource managment. Chapter 3 describes how a model-based resource provisioning process works in detail. In Chapter 4, a performance model of Quake game application is presented, while Chapter 5 presents a heuristic of resource allocator for multiple services. The details of a simulation study and the results of model-based service provisioning are presented in Chapter 6. Finally, Chapter 7 concludes this thesis. Appendix A gives a list of abbreviations used in this thesis.

Chapter 2

Related Work

The first section discusses some projects that have a direct connection with, or share the vision of, adaptive resource management to solve the problem mentioned in Chapter 1. Because my work is applied to the FPS genre of online game, the final section of this chapter talks about some existing online game platform approaches for the resource management.

2.1 Adaptive Resource Management

A number of research efforts have been put forward in addressing various issues of resource management for various computing environment. The objective of all research efforts is to solve the problem on how to allocate resource efficiently for task under any circumstance. And thus tasks can be served decently with enough resource. The following sections describe some of the research works that try to implement resource management infrastructures.

2.1.1 Resource Management for Single Server

Welsh et al. ([10]) propose a new design framework for highly concurrent server applications, which is called the *staged event-driven architecture* (SEDA). It is intended to support massive concurrency demands and simplify the construction of well-conditioned services. SEDA combines aspects of threads and event-based programming models to manage the concurrency, I/O, scheduling, and resource management needs of Internet services. In SEDA, Applications are constructed as a network of stages, each with an associated incoming event queue. Each stage represents a robust building block that may be individually conditioned to load by thresholding or filtering its event queue. In addition, making event queues explicit allows applications to make informed scheduling and resource-management decisions, such as reordering, filtering or aggregation of requests. SEDA makes use of dynamic resource throttling to control the allocation and scheduling of application components, allowing the system to adapt to overload conditions. This architecture allows services to be well-conditioned to load, preventing resources from being over-committed when demand exceeds service capacity. SEDA emphasizes request admission control to degrade gracefully in overload. A utility can avoid overload by expanding resource slices and recruiting additional servers as traffic increases, but it only reactively provisions CPU resources (by varying the number of threads) across multiple service stages within a single server.

One point SEDA does not address, however, is performance isolation for shared servers which can be implemented by my model-based approach. In my simulation, VC slices provide performance isolation and enable the utility to use its resources efficiently. The VC slices are chosen to allow each hosted service to meet service quality targets (such as response time) negotiated in service level agreements (SLAs) with the utility. And VC slices vary dynamically to respond to changes in load and resource status.

2.1.2 Resource Management for Cluster Utility

IBM's Océano project [11] incorporates management of SLA into its system to use a simple event driven system to decide the resource allocation. Its similarity to my VC resource management concept is that it includes two tiers: a large pool of "dolphin" servers that serve as protection layer for the system while demand is in spike and a fixed small pool of "whale" servers. Whale servers are permanently allocated to services to guarantee the minimum level of agreed QoS, while "dolphin" servers are dynamically included into and expelled from the service-specific server pools depending on the load conditions of the services. SLAs are observed by event driven mechanisms. But the difference from my VC resource management algorithm is that VC spans multiple geographical locations and VC resource pre-allocation is regarded as a *facility location problem* (FLP) based process which deals with capability and capacities in terms of the applications.

The more recent Neptune [12] work proposes an alternative to provisioning (or partitioning) cluster resources. Neptune maximizes an abstract SLA yield metric and each server schedules requests locally to maximize per-request yield. Neptune has no explicit provisioning; it distributes requests for all services evenly across all servers, and relies on local schedulers to maximize global yield. While this approach is simple and fully decentralized, it precludes partitioning the cluster for software heterogeneity which can be implemented by my model-based approach integrated with PCU environment. My performance model can be customized for heterogeneous games with the available benchmarks for measuring the performance of commercial game servers and then used separately to partition the VCs.

2.1.3 Model-Based Resource Management

Modeling system behavior is a good way to understand application and enable the system to achieve important resource management goals. Many studies have been performed to understand the system model and its impact on the Internet.

For the streaming media application, Cherkasova et al. [8] introduce a set of basic benchmarks to measure the basic capacities of streaming media systems and reveals the main system/application bottlenecks and properties. The set of basic benchmarks allows one to derive the scaling rules of server capacity for media files encoded at different bit rates and delivered from memory versus disk. Using experimental testbed, it shows that these scaling rules are non-trivial. It designs a performance model of media server which uses the basic benchmark measurements for estimating multiple resource requirements of a particular stream in order to compute the server capacity for delivering a realistic workload. But it fails to combine the proposed media server performance model with workload analysis output to design the admission control strategies. The model-based admission control infrastructure can be expected to prove that a streaming media service can efficiently determine whether sufficient CPU, memory and disk resources will be available for the lifetime of a particular request as a function of the requests already being delivered from the server. In my work, my performance model is integrated with an admission control scheme with content simulation results. With my resource management system, given the real-time requirements of each client, a server has the ability to employ admission control algorithms to decide whether a new client request can be admitted without violating the quality of service requirements of the already accepted requests.

For the web service utility, Doyle et al. [7] demonstrate the potential of MBRP for resource management. It shows how MBRP policies can adapt to resource availability, request traffic and service quality targets. These policies leverage the models to predict the performance effects of candidate resource allotments, creating a basis for informed, policy-driven resource allocation. Their positive experimental results illustrate that MBRP is a powerful way to deal with complex resource management challenges, which enables the system to achieve differentiated service quality, performance isolation and storage aware caching.

Depending on the game genre, there are many different internal methodologies to run the services from normal web or media service. Abdelkhalek et al. [13] mention that both client and server side processing in FPS online games are based on "round". That is to say, when a client joins a game session it enters a loop performing the following actions in well defined iterations: (i) get the player input and send it to the server, (ii) receive server updates for all relevant entities for the request the client sent in the previous time slot, (iii) perform prediction on all entities (e.g., players and objects with associated actions), and (iv) render the next frame. On the server side, upon receiving client input, the server determines how this interacts with the rest of the virtual world in a compute intensive task. The server replies only to explicit client requests, assuming that clients are always active sending frequent requests with user actions. Ideally, the server replies with updates to client requests within the same client time slot that the client sent the request. Each round may last several hundred milliseconds depending on the game genre. This specific characteristic means that the established models for Web service behavior can not be applied to generate the performance model for game service. Another reason why the game service can not use the existing models is that requests to static web objects follows a Zipf-like popularity distribution mentioned in [14, 15] but traffic distribution to the game service depends much on the game features mentioned in [16, 17, 18].

For most Internet applications, server performance can be modeled by the arrival rate proposed in [19]. Ye and Cheng ([20]) choose this feature to build performance model for MMORPGs and demonstrate that the performance metrics at the server side have a strong linear relationship with the number of concurrent players. With these results service providers can reasonably predict resource requirements for their gaming infrastructure at runtime in an automated way. Ye and Cheng's effort has not considered the effect of an FPS game's server side bot, which plays automatically and intelligently without human control and brings more overload to the server resource than human player. It also does not present validation for the proposed model for MMORPG game.

My proposed performance model is generated based on the FPS game server to meet its specific characteristics. And then it is validated and then merged with the computing utility resource management algorithm. With the help of this performance model, FPS game service provider could benefit from the game which requests the necessary resources to meet variable demands. The simulation result suggests that this integration can be applied for other automated

IT resource allocations at runtime in the context of on-demand utility computing systems.

2.2 Online game Platform Related Projects

An on-demand service platform for hosting large-scale multiplayer games is proposed in [21]. The proposed idea is sharing IT resources across multiple game titles or customers by dynamically provisioning and deprovisioning resources for a title or customer from a shared resource pool. This pool has IBM Tivoli Intelligent Orchestrator (TIO) as a fully dynamic resource allocation scheme that helps allocate resources with the information of the CPU utilization on the active game server. If the utilization crosses a threshold, TIO installs the appropriate game server software on a new idle server and adds it to the active set. TIO then executes a custom workflow that notifies the redirection server that a new game server is available for joining players. Similarly, when players leave a game server and it becomes idle, TIO removes it from the active set and returns it to the idle pool after all players have disconnected. The redirection server is an instance of a game server with a slight modification to the communication protocol to allow clients to first issue a server assignment query and receive a redirect response to connect to the appropriate server, rather than connecting directly. This one-by-one server addition/subtraction makes this on-demand platform more dynamic than the VC mentioned in [9]. Although its performance is found to be stable over a range of loading conditions, it is less cost-effective since it can not avoid the disadvantage of frequent reallocations during transient fluctuation, game performance or server utilization. This resource allocation algorithm in [21] was not compared with mine, but my simulation results show that the VC holds its metric steady during the scaled simulation time and [9] already proved that resource utilization in VC outperforms the one in Service Grid dynamic algorithm presented in [22] with underloaded or nominally loaded conditions.

Deen [23] successfully uses IBM OptimalGrid middleware to build up an on-demand platform for MMORPGs. IBM OptimalGrid technology is research middleware designed to hide the complexity of creating, managing and running large-scale parallel applications on any kind of heterogeneous computational grid. OptimalGrid was created, in particular, to address scientific and technical computing problems that are parallel and connected, so the OptimalGrid object model is general enough to handle a wide variety of other coupled parallel applications, including MMORPGs. OptimalGrid automates the task of resource allocation on a computational utility or grid to optimize the performance of running applications. It efficiently and automatically partitions a given problem throughout a large collection of computers. OptimalGrids runtime model uses three different services that are placed on computers in the grid: autonomic program manager (APM), computer agents (CAs) and TSpaces servers. The APM oversees the execution of the application and contains any load-balancing and global-scheduling control used by an application. The APM directs the work assigned to and executed by each of the CAs, which has a variable problem partition (VPP), that is, the set of work units assigned to the CA. The OptimalGrid object model assumes that an application can be described as a graph where the nodes on the graph contain data, methods and pointers to neighbors. In OptimalGrid terminology, these nodes are called original problem cells (OPCs). OPCs are the "atomic" problem units (the smallest pieces) of a problem that represents a unit of computation. In general, OPCs interact with their neighbors, sharing information to produce a larger, big-picture computation. Therefore, an OPC must communicate its state with its neighboring OPCs. During load time, OPCs are grouped into collections, which are precomputed and have predefined dependencies on other collections. Each computational node is assigned one or more OPC collections. This defines both the computational workload and the communication workload (the collection edges) for that node. The VPP for each CA is assigned zero or more OPC collections, the actual number of which is changeable through a load-balancing operation by the APM. Thus, a computer runs a CA that holds a VPP, which in turn holds zero or more OPC collections that are each made up of one or more OPCs.

Given these considerations, [23] consider the map in Quake II as the entry point for the OPC creation. Because the positions and orientations of walls and other surfaces in the map are used to create a binary space partitioning (BSP) tree, it divides the entire map into a number of small, irregular convex polyhedrons, each of which corresponds to a "leaf" in the tree. A typical leaf collection may be the size of a portion of a room on the map. Therefore, the BSP tree leaf collections of a Quake II map is chosen as the OPCs. This solution may not be the best to apply to different game engines that face different resource limitations when required to host large numbers of players. The ability to partition and distribute the computational needs of these alternate engines would require analysis of the particular design and behavior of the engine to make the appropriate partitioning decision. This feature makes the OptimalGrid solution not capable of hosting several different games at the same time. This limitation can be solved in my multiple VCs solution, which can host different services with different design features and can easily be customized and generalized.

Chapter 3

Model-Based Resource Provisioning

A *computing utility* (CU) aggregates computing power from various owners and distributes it to the subscribers. The computing power from a CU can be used to construct a web-server cluster, to perform distributed computing or to setup a storage area network. The hosted services can impose varying loads on the infrastructure to respond to changes in workload demand and resource status. In order to manage hosted services efficiently, we need to answer these questions: (i) What is the available capacity? (ii) What capacity is required to service user requests without degradation of quality? *Model-based resource provisioning* (MBRP) mechanism can be used to answer these questions because the applications hosted on CUs have predictable resource demand and then the resource demands could be a function of load with stable, observable characteristics.

MBRP is an adaptive resource management technique that departs from traditional ones using reactive heuristics with limited assumptions about application behavior. The basis of MBRP is that internal model capturing service workload and behaviour can enable the utility to predict the effect of resource choices on application service quality and workload intensity. This performance model needs to continuously monitor load and plan resource allotments by estimating the value of their effects on service performance. MBRP can address the provisioning problem of how the service automatically requests the necessary resources from the infrastructure to meet *service level agreement* (SLA) targets at its projected load level. To conclude, MBRP can achieve the following goals quite easily:

- *On-the-fly server capacity prediction:* The models can determine aggregated resource requirements for all hosted services at observed load levels. This is useful for admission control or to vary hosting capacity with offered load.

- *Differentiated service quality:* Because the system can predict the effects of resource allocation on service quality, it can plan efficient resource allotment to SLA targets.
- *VC Resource management for diverse services:* Each game has a distinct CPU-resource request characteristic. This model can be applied to capture these properties, enabling the resource manager to pick up allotment to meet performance goals.

The flexibility and diversity of MBRP make it a powerful tool for interactive resource management mechanism in *public computing utilities* (PCUs).

In this thesis, the potential of a MBRP for resource management is demonstrated. Figure 3.1 shows a block diagram of an adaptive resource management process that is typically necessary to deploy online games on PCUs.

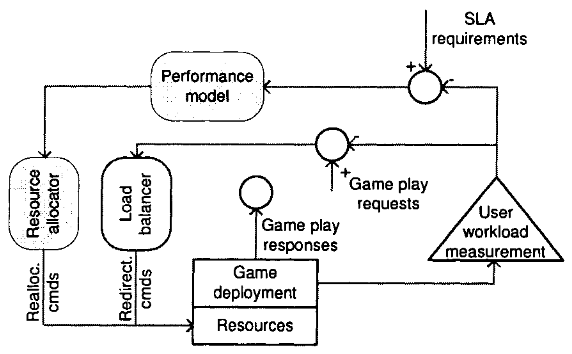

Fig. 3.1 Adaptive resource management for mapping online games on PCUs

In our scheme, each game deployment is allocated a VC. VCs are service specific collections of resources with minimized allocated resources cost and maximized the capability of the PCU system to provide better services. A VC can include server slices from geographically distributed servers. Figure 3.1 shows the major modules necessary to implement the resource management process. The user workload measurement module estimates the number of active player sessions in the game. Long-term and short-term averages of this estimate are used by the resource management process. The short-term average is used for redirecting the requests based on the predicted service statuses among the servers that are already part of a VC. In addition to the server load, the redirections consider proximity. For instance, if multiple replicas of the game are available, the closest (in terms of network distance) is chosen to map the player session. The long-term load average is used for reallocating resources to the VC. The resource allocation process dedicates some resources from the common pool held by the PCU to the particular VC.

Because the resource allocation is a competitive process among diverse applications it is necessary to resolve the workloads from game specific units (e.g., number of active player sessions) to resource units. This conversion is performed by the *performance model* module. The *resource allocator* module takes the workload expressed in terms of server units and performs the reallocation exercise which incrementally changes the VC configuration. This incremental change in VC configuration takes the SLA requirements into consideration as well.

In this thesis, the performance model and resource allocator modules are evaluated. The performance model presented is derived for Quake – a popular online game. It converts the game workload into server capacity requirements. For this work, only server CPU utilization is measured. We use this performance model to evaluate a resource allocation heuristic.

Chapter 4

Performance Model for an Online Game

Multiplayer online games have become a very popular form of multimedia service on the Internet, and popular *massively multiplayer online role-playing* (MMORPG) game titles have large numbers of subscribers shown in Figure 4.1 up to July 2006. Among them, the class of games known as FPS has dominated much of today's large-scale, highly interactive virtual gaming industry. In virtually all cases, multiplayer games are enabled by a central server. Clients connect to this server, which is responsible for interpreting their actions, maintaining consistency and passing information among them. These kinds of traditional games rely on a rigid centralized-server approach, causing games to become prone to performance bottlenecks and downtime. I intend to create an infrastructure that provides a superior game playing experience. Using on-demand grid technology as the underlying infrastructure for hosting online games is a good way to solve the provision and demand problem, which is proposed in [21] and [24]. These grid based systems can provide a flexible, cost-effective platform to support the hosting of applications for Internet services, but a key challenge for them is to enable the design of a "utility-aware" game whose capacity scales with the service demand to maintain the quality target at the least cost.

To achieve such a design that a game which automatically requests the necessary resources from the computing utility infrastructure adapts to variable demand, the capacity of game servers is measured in order to evaluate the amount of available system resources for admitting new client requests. In this work, a load-aware performance model was devised for online game applications to predict the necessary values of the candidate resource allotments under changing loads.

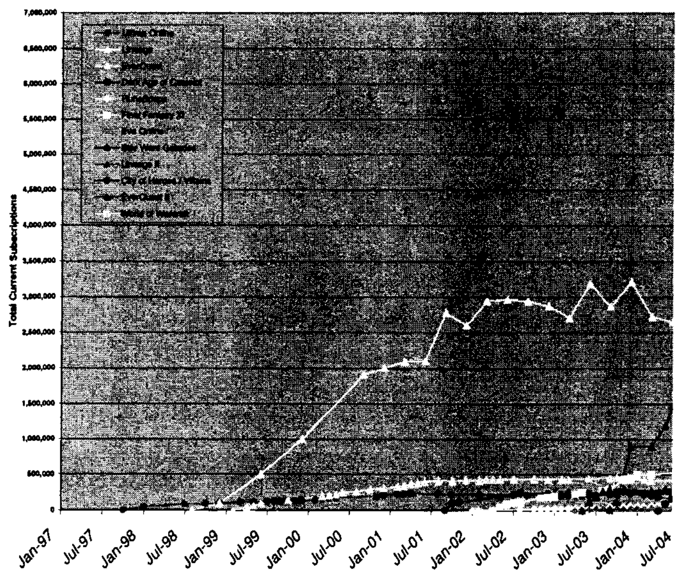

Fig. 4.1 MMOG active subscriptions

4.1 Introduction of Server Side Bot and Client Side Bot

Normally the game server works by accepting the human player connections from various locations to run on itself, but when the game server is short of human players, or there is need to control participation for testing and debugging purposes, the Quake engine has a built-in server side bot (S/S bot) [25] that plays just like a human opponent. The S/S bot can move like a real player, uses all the weapons available to the player and features the best level navigation - it knows where to go and how to win the game. With the release of the Quake I source code, bot authors can finally add bots directly to the game code. Alternatively, the S/S bot can also be coded in QuakeC, which is a scripting language for Quake engine based games [26] to modify and organize players, monsters, buttons and weapons in the Quake world. With complicated AI knowledge coded, the S/S bot can be a challenging opponent against the human player. In my experimentation, the server side bot FrikBotX [27] coded in QuakeC is used. FrikBotX is a classic bot for the game Quake and it interfaces with the game engine as a real client world.

A client side bot has a similar purpose to a server side bot, but it plays by connecting to the game server from another machine, just like a human player. In order to make the benchmarking procedure automated, the human player was replaced by Mikebot, an optimized client side bot [28]. By replacing the human player with automated players client traffic can be created actually.

Due to the lack of human participation in my experimentation, a demo-recording tool is used to record the keyboard and mouse events for later playback. The game engine was then modified to obtain user input from this previously recorded file. Thus, there is no need for the intervention of human players. From my collected data, the recorded file can generate almost the same workload as my client side bot. Note that the goal of the recorded file was not to accurately recreate real player movements and actions; rather, they were meant to create load on the server to compare with the client side bot. In order to remove the effect of different map complexity, the same map type - *deathmatch* (dm) is chosen in my experimentation, so the map size on disk in KB can simply represent the map complexity in this scenario. Figure 4.2 compares the overload brought by Mikebot and a real human player and shows no significant difference in CPU utilization. My experimentation proves that Mikebot can emulate the human player and the gameplay is still very interactive and logical with the client-side bot. Therefore, Mikebot can be used to automate the benchmarking procedure.

Accordingly, a simple set of basic benchmarks is introduced to analyze the main bottlenecks of this game server to build this model:

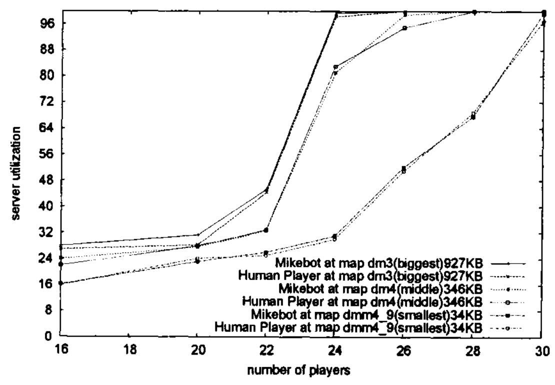

Fig. 4.2 Comparison between MikeBot and human player on server utilization.

1. *Server-side (S/S) bot benchmark:* measures game server capacity when all the players are bots playing on the server side.
2. *Client-side (C/S) bot benchmark:* measures game server capacity when all the players are bots, simulating human players on client side.

Twilight, derived from the Quake engine, is chosen as the game server since it is an open source based application that can run on several platforms (Win32, Linux, MacOS, etc.). Its map format is compatible with the one of Quake I and Quake II and makes much more use of the graphics hardware, using vertex programming adds many visual improvements and makes the game run faster. It is also one Quake engine that can support my C/S and S/S bots well together.

4.2 Set of Basic Benchmarks Experimentation

With these benchmarks, a server overload metric is used to determine whether a server has reached its maximum capacity for the applied workload. This metric is defined as *server capacity saturation percentage (L)*, where 1 equals the game server CPU consumption at 100% (can not accept any more player requests). S/S benchmark is a completely automated benchmark which runs a sequence of tests with an increasing number of S/S bots. During each test, two performance data are collected: (a) the size of the map and (b) the CPU usage percentage. In order to make sure that none of the bots remain in the server's buffer cache (memory), each test is launched from the beginning point every time.

During the game play, besides the S/S bot interaction itself, the game map complexity is another effect to be considered. The complexity of the map of the same type is defined by its layout and the number of objects it includes. In order to ignore the type effect of different map to make my performance model easier to be investigated, the same type (dm) is also chosen to run my S/S bot benchmark. The main goal of S/S bot benchmark is to determine the number of S/S bots that can make the game server saturated under a specific map. Figure 4.3 shows that the smaller the map is, the smaller number of bots the game server can support. That is to say, the maximum achievable number of bots depends on the size of the map. The induced higher level of interaction among players in the smaller map with less complexity makes the server run computation more fiercely and easily reach its saturation proximity.

An observation from Figure 4.3 is that the game server capacity increases with map complexity, but in different scale. Increase the complexity by a factor of 27.26 (from 34KB to 927KB), the number of S/S bot supported by the game server only increases by a factor of 2 (from 12 to 24). Figure 4.4 illustrates that in more detail. It also displays non-linear relationship between map complexity and the number of S/S bots supported. The game server experiences more overload when handling a larger number of S/S bots in a map of less complexity than handling a smaller number of S/S bots in a map of more complexity. Under the S/S bots benchmark, the game server

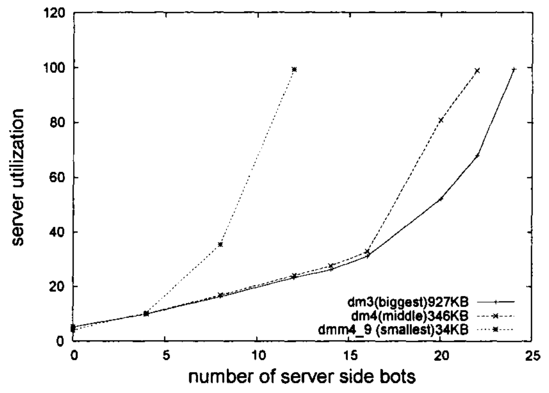

Fig. 4.3 Server utilization under server-side bot benchmark.

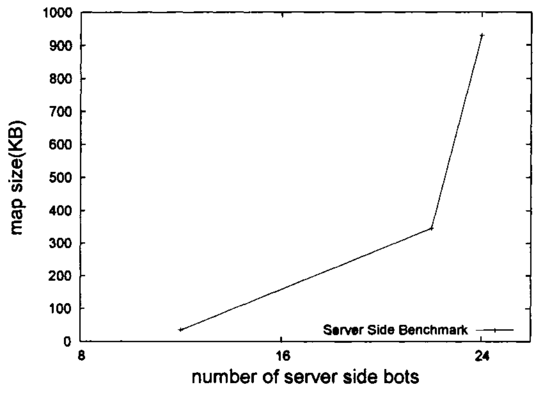

Fig. 4.4 Server-side bot benchmark.

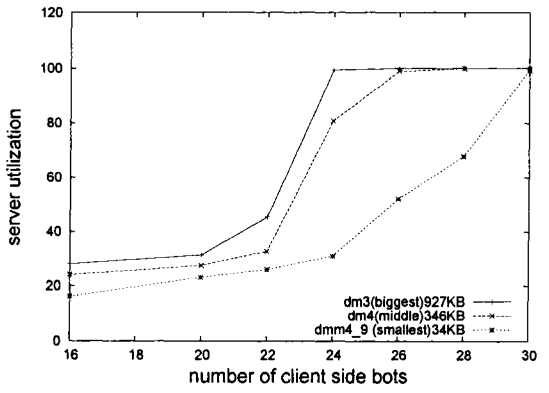

Fig. 4.5 Server utilization under client-side bot benchmark.

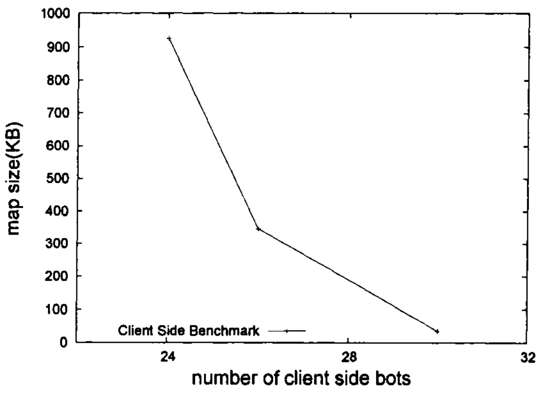

Fig. 4.6 Client-side bot benchmark.

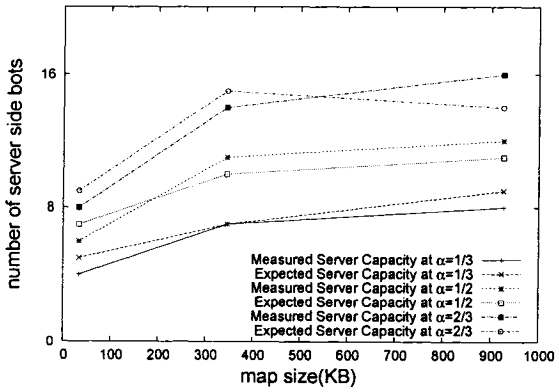

Fig. 4.7 Measured vs expected server capacity.

is CPU bounded: CPU usage percentage reaches 100%. It is the main resource limiting the server performance.

C/S bot benchmark measures the game server capacity in Figure 4.5, when each client side bot accesses the game server. Similar to the S/S bots benchmark, the C/S bot benchmark is completely automated using all three maps of the same type and runs a sequence of tests with an increasing number of clients, where those clients run on one machine. Figure 4.2 shows that the C/S bot can match the human player behavior and won't introduce much difference on the client side.

Figure 4.6 shows the maximum capacity of C/S bots achievable by the game server across three maps of different complexities in the C/S bots benchmark. First, the larger map needs less clients to saturate the game server compared to the smaller map. This is because the larger map has the information of more objects and much more elaborate layout in the packet sent to the game server, which makes the processing time at the server increase a lot. As a consequence of the increased server processing time and the size of the incoming request queue, server response rate drop and the number of C/S bots the server can hold properly drops as well.

Secondly, the curve in Figure 4.6 follows the opposite direction compared to the S/S bots curve in Figure 4.4. In the smallest map (dmm4_9), 12 S/S bots can saturate the server. However, in the same map the game server is saturated with 30 C/S bots. That means one C/S bot brings less overload to the server compared to one S/S bot, and one S/S bot brings the higher CPU utilization with memory access in the game server compared to one C/S bot, which brings communication and computation cost with network access. The S/S bot, coded by complicated AI knowledge and stored entirely in memory and interpreted by the engine, works via artificial intelligence routines pre-programmed to suit the game map, game rules, game type and other parameters unique to the game. The game engine takes much time for analysis or prediction and then computes the response for the constantly changing condition between S/S bots. But C/S bots only send the packet request to the game engine, which processes it with some preprogrammed routines and then sends out a response, causing the C/S bot to perform its own computation of this response packet without hogging server resource. In this way, the game engine can eliminate unnecessary analysis time to be in a less resource intensive condition compared to the one brought by S/S bot.

4.3 Game-Aware Performance Model of Game Server Capacity

In this section, a game-aware performance model is presented for the game server capacity based on a cost function derived from the set of basic benchmark measurements explained in Section 4.2. This cost function defines a fraction of the system resources needed to support a particular client as a function of the map complexity. Because the CPU utilization is caused by memory and network access type, the cost is matched with a resource access type with S/S or C/S bots that accesses the game server. I categorize the resource access into two types:

- *Memory access:* if the bot exists on the server side, it introduces only computation overload at the game server with code stored in memory running in CPU.
- *Network access:* if the bot connects to the server through the network resource, it introduces both computation and communication overload at the game server.

The following notations are used in this thesis:

- $N^{s}_{M_i}$ – The maximum measured S/S bot number under the S/S bots benchmark at a map M_i once the server is saturated
- $N^{c}_{M_i}$ – the maximum measured C/S bot number under the C/S bots benchmark at a map M_i once the server is saturated
- $N^{cs}_{M_i}$ – The current measured S/S bot number under the S/S bots benchmark at a map M_i
- $N^{cc}_{M_i}$ – the current measured C/S bot number under the C/S bots benchmark at a map M_i
- $C^{m}_{M_i}$ – a value of cost function for a bot with memory access at a map M_i
- $C^{n}_{M_i}$ – a value of cost function for a bot with network access at a map M_i

Under the S/S bots, all the bots have a memory access type because all bots exist on the server side, and hence each bot requires a fraction of server capacity defined by the $C^{m}_{M_i}$. Under the C/S bots, all the bots have a network access type and requires a fraction of server capacity defined by the $C^{n}_{M_i}$.

The following capacity equation describes the maximum server capacity measured under a set of basic benchmarks for each map complexity, where 1 indicates the state of server being saturated.

$$N^{s}_{M_i}C^{m}_{M_i} = 1 \tag{4.1}$$

$$N^{c}_{M_i}C^{n}_{M_i} = 1 \tag{4.2}$$

By solving these equations, the cost function values can be derived

$$C^{m}_{M_i} = \frac{1}{N^{s}_{M_i}} \tag{4.3}$$

$$C^{n}_{M_i} = \frac{1}{N^{c}_{M_i}} \tag{4.4}$$

Let L be a current workload as server loading metric presented by a game server, where

$$L = N^{cs}_{M_i}C^{m}_{M_i} + N^{cc}_{M_i}C^{n}_{M_i} \tag{4.5}$$

This model is cheap to evaluate the server performance and it captures the key behaviors that determine application performance. This was originally developed to improve the understanding of service behavior and to aid in static design of server infrastructure. Because it predicts how resource demands change as a function of offered load, it can help act as a basis for the dynamic provisioning in a shared hosting utility. At any given time, it is impossible to run two maps on one game server, so the above equation can be simplified as:

$$L = N^{cs}_{M}C^{m}_{M} + N^{cc}_{M}C^{n}_{M} \tag{4.6}$$

If $L \geq 1$ then the game server is overloaded and its capacity is exceeded. Otherwise, the game server operates within its capacity.

4.4 Performance Model Validation

This performance model of game server is validated by comparing the predicted and measured server capacity for a set of synthetic workloads.

Let the number of bots from an outside connection and the number of bots from within the server be defined as the following:

- $\alpha N^{s}_{M_i}$, where $\alpha \leq 1$ and $N^{s}_{M_i}$ is the measured server capacity under the S/S bot benchmark

- $(1-\alpha)N^c_{M_i}$, where $N^c_{M_i}$ is the measured server capacity under the C/S bot benchmark

If my performance model of server capacity is correct, then under the set of synthetic workload the server maximum capacity should be reached.

The mix workload is ran for $\alpha = \frac{1}{3}, \frac{1}{2}$, and $\frac{2}{3}$. In these workloads, the S/S bots is fixed according to a formula defined above and slightly increased the number of the C/S bots to determine when the server capacity is reached. Figure 4.7 shows the variation of the measured capacity with the expected server capacity, where it can be observed that the measured server capacity closely follows the expected capacity for different values of α.

Chapter 5

Resource Provisioning Algorithms for Multiple Services

This chapter develops an optimization model for the creation process of multiple virtual clusters on the basis of the mathematical model proposed in [9]. This model is aimed to achieve the best resource utilization and mathematically reduce the entire system cost. Section 5.1 introduces the basic components in the public computing utility and the resource allocation for a single service based on this infrastructure. In Section 5.2 the mathematical model is presented for resource allocation of multiple services and analyzes the practical validity of such model. The solution methods of the developed model is analyzed and a heuristic that solves this model is explained in Section 5.3. Some contributions in my heuristic algorithm are also mentioned in the end of Section 5.3. An example of this heuristic applied is illustrated in Section 5.4

5.1 Resource Allocation for Single Service

A *public computing utility* (PCU) connects resources belonging to various organizations into a single system such that clients can get necessary resources for their computing needs in an on-demand basis and has the CU's features mentioned in the beginning of Chapter 3. Therefore, the PCU model is proposed as an infrastructure to host various wide-area services considering service resource cost and QoS aspects and then allocates partitions of these resources to different hosted applications. These resource partitions are called virtual clusters (VCs). Another object in this PCU model is the anchor point (AP), which represents centroids of the demand distribution in a geographical area to provide us an abstraction of service request intensity in a network. AP has several attributes such as location, demand intensity and type. A VC has to be designed with sufficient resources allocated from the big resource pool of a PCU such that the APs are covered at or above a predefined level of service. An efficient and fast solution to solve the resource allocation for single service is already presented in [9].

The problem of resource allocation to create a VC is regarded as a *facility location problem* (FLP) [29]. In general, a FLP is concerned with optimally placing a number of facilities (such as serving resources) to cover demands at a predefined set of points (such as APs). When a server is initially mapped into a VC, the application or operating system needs to be installed and then the initial data should be loaded from the appropriate data sources. This implies a fixed cost for mapping a server into a VC. Servers that are part of a given VC have a fixed capacity to handle a fixed number of concurrent requests. Therefore, the capacity of the serving nodes should be considered while allocating the serving resources to cover the demand. So the VC creation is further regarded as a *capacitated fixed-charge location problem* (CFCLP). The CFCLP assumes that placing a facility at a given candidate location incurs a fixed cost and each facility has a limited capacity of covering the demands. Further, the serving resources bound by the VC creation process to a particular AP have to provide the best coverage for the demand originating from that AP. This is achieved by minimizing the *covering cost* in a *capacitated fixed-charge location problem* (CFCLP) model equation 5.1.

Parameter	Notation
y_{ij}	network delay between AP$_i$ and server$_j$
b_{ij}	bandwidth between AP$_i$ and server j
k_j	capacity of server$_j$ that denotes the maximum number of sessions it can handle
c_{ij}	cost of covering AP$_i$ by server j
h_i	demand of AP$_i$ that denotes the number of created sessions
f_j	fixed cost for allocating node j as a facility

Table 5.1 Single VC network notation

The PCU system can be imagined as a graph with the servers and APs as the nodes and the network links as the edges. Let S be the set of potential server nodes and V_a be the set of APs. In addition, we define the following parameters in Table 5.1. In the PCU network, the lower bandwidth can increase the cost of providing better services over the network traffic to cover the

AP demand, on the contrary, the higher delay can increase the cost. Therefore the covering cost c_{ij} is defined as $c_{ij} = \left(\alpha y_{ij} + \frac{\beta}{b_{ij}}\right)$ where α and β are VC specific. And the QoS of a service is also decided by the network bandwidth and the delay since in most cases the QoS is represented by the response time and the throughput. To be concluded, the covering cost c_{ij} can be considered as the inverse of the delivered QoS.
The decision variables defined in [9] listed as follows:

$$s_j = \begin{cases} 1 & \text{if server}_j \text{ is part of the VC} \\ 0 & \text{otherwise} \end{cases}$$
$$u_{ij} = \begin{cases} 1 & \text{if server } j \in S \text{ covers AP } i \in V_a \\ 0 & \text{otherwise} \end{cases}$$

With these parameters, the optimization problem for creating a VC becomes (with $i \in V_a$ and $j,k \in S$):

$$\text{minimize} \quad \sum_j f_j s_j + \sum_i \sum_j h_i c_{ij} u_{ij} \tag{5.1}$$
Subject to
$$\sum_j u_{ij} \geq 1 \quad \forall i \tag{5.2}$$
$$\sum_i h_i u_{ij} \leq k_j s_j \quad \forall j \tag{5.3}$$
$$\tag{5.4}$$

The objective function given in (5.1) tries to minimize the number of servers (with s_j) in the VC while maximizing the delivered QoS (with c_{ij}). Using demand-weighted cost ($h_i c_{ij}$) places the server closer to the APs with high demand.

The Constraint (5.2) guarantee that every AP is covered. The Constraint (5.3) makes sure that each server is serving under its capacity. Here only some essential formulation constraints are listed.

A centralized heuristic algorithm to solve this formulation is developed in [9]. This algorithm is based on the drop heuristic given in [29], using a local search technique [30]. It works in two phases: firstly, one feasible configuration is found for a VC that includes all available servers and each AP is greedily assigned to a server minimizing the covering cost c_{ij}. Once the feasible allocation is found, the second phase is initiated to predrop each of the already selected servers to find out the servers, which can produce the largest reduction for objective function 5.1 and can be removed without hurting other APs allocation.

The static content of this algorithm is that it can choose the best server resource greedily to cover the AP demand considering to minimize the value of objective function. This algorithm can be applied to other Internet single applications though the simulation study in [9] only considers document retrieval service. But this solution heuristic is a centralized algorithm, which makes it infeasible to scale to a large infrastructure.

5.2 Resource Allocation for Multiple Services

Resource allocation for multiple services is different from the allocation for single service. We need to take into account several aspects (i) the fairness on allocating resources to each VC based on the demand from AP group side. That means each VC has the equal right to get the relatively best resource from the public resource pool without following the order by some human-defined priority. (ii) The system-wide resource utilization, which means not only the resource utilization in each VC, but also the entire PCU has the resource utilization as maximized as possible. (iii) Another point to be considered is that the entire cost in the PCU can be minimized as much as possible and QoS for all the VCs can be maximized instead of for any individual VC.

Considering the above conditions, we again model the creation of multiple VCs as an optimization problem that tries to allocate facilities to demand points while maximizing the total preference, g_{ij} which is explained in Table 5.2. For the creation of multiple VCs, the model is very similar to the one for the creation of single VC. Therefore, the model have many same ideas as the above one in Section 5.1, such as facilities, demand points, QoS metric and demand allocation cost. But the difference is that demand point is considered as AP group during the creation of multiple VCs. We group APs with demand for the same service, so for the purpose of multiple VCs creation, we have multiple AP groups to be considered as demand point. Also they share some assumption like a fixed cost for using each facility. Obviously, the multiple VC problem can also be modeled as allocation of resources incur a fixed cost (resource rental) and capacity of each resource is always limited.

The PCU system is modelled as a graph with the servers and AP groups as the nodes and the network links as the edges. Let S be the set of potential server nodes and V be the set of AP groups. In addition, the following parameters are defined in Table 5.2. The covering cost c_{ij} is defined as $c_{ij} = \left(\alpha y_{ij} + \frac{\beta}{b_{ij}}\right)$ where α and β are VC specific.

Notation	Description ($i \in V$ and $j \in S$)
h_i	demand at AP_i group
k_j	the server$_j$ capacity that denotes the maximum number of concurrent sessions it can handle
c_{ij}	the sum of covering cost between each member in the AP group i and server$_j$
f_j	fixed cost for allocating node j as a facility
y_{ij}	the sum of network delay between each member in the AP group i and server$_j$
b_{ij}	the sum of bandwidth between each member in the AP group i and server$_j$
p_{ij}	sufferage preference of allocating server$_j$ to AP group i detailed in Section 5.3.1
t_{ij}	deficiency preference of allocating server$_j$ to AP group i detailed in Section 5.3.2
g_{ij}	$g_{ij} = t_{ij} + p_{ij}$

Table 5.2 Multiple VCs network notation

Here the S is divided into several VCs, each one is represented by VC$_i$ ($i \in V$). The decision variables are defined as follows:

$$s_j = \begin{cases} 1 & \text{if server } j \text{ is part of the VC}_i \\ 0 & \text{otherwise} \end{cases}$$

$$u_{ij} = \begin{cases} 1 & \text{if server } j \in S \text{ covers AP group } i \in V \\ 0 & \text{otherwise} \end{cases}$$

With these parameters, the optimization problem for creating multiple VCs becomes (with $i \in V$

and $j, k \in S$):

$$\text{maximize} \quad \sum_j \frac{s_j}{f_j} + \sum_i \sum_j g_{ij} u_{ij} \tag{5.5}$$

Subject to

$$\tag{5.6}$$

$$\begin{aligned} u_{ij} &\leq s_j \quad \forall i, j && (5.7) \\ \sum_j u_{ij} &\geq 1 \quad \forall i && (5.8) \\ s_j &= 0, 1 && (5.9) \\ u_{ij} &= 0, 1 && (5.10) \end{aligned}$$

The objective function given by Equation 5.5 has two parts: first part is based on the number of servers in the VC and the second part is the total preference value g_{ij}. By maximizing the objective function, the first part reduces the total priming cost of the VC, while the second part increasing the allocation preference value and thus increasing delivered QoS. The preference value given in the second part is used to place the server closer to the AP group which has the bigger sum of sufferage preference value and deficiency preference value.

The intra-VC bandwidth requirement is defined by the Constraint (5.6). The Constraint (5.7) denotes that a server can cover an AP group only if it is part of VC. The Constraint (5.8) makes sure that every AP group is covered.

The above formulation maximizes the delivered QoS based on the sufferage preference value and maximized the resource utilization in the entire PCU system based on the deficiency value. But it does not guarantee a given QoS level. This becomes an issue when a value for QoS is agreed upon on the SLA.

5.3 Heuristic for Solving Resource Allocation for Multiple Services

Before this new algorithm is developed – *optimized mapping algorithm* (OMA) to multiple services resource allocation, an alternative way to solve this problem is also considered, just iteratively run the resource allocation mechanism for single service mentioned in [9] for each AP group. First, the suitable servers are chosen from the big resource pool to create one VC for the first AP group and then run this algorithm again with the rest of servers to create another VC for the second AP until all VC are created or all the servers are picked up. But there is some problems behind this iterative solution. (i) It requires to consider the different order to create VC for AP group with some defined priority (ii) This algorithm is not run simultaneously to create each VC fairly enough. (iii) Though every time allocation can find the best server choice to create one VC, it can not guarantee that the final allocation for the multiple VCs system is the best in terms of QoS. (iv) The solution doesn't consider the balance of demand and provisioning when allocating resource to create VC. My current heuristic solution is going to solve these problems, which defines the allocation matrix with sufferage preference algorithm in Section 5.3.1 and deficiency preference algorithm in Section 5.3.2 to create multiple VCs in the PCU.

5.3.1 Sufferage Preference Algorithm

Sufferage preference algorithm is a concept of resource scheduling heuristic, which is already proved in [31] to be the best batch mode heuristic compared to Min-min heuristic and Max-min heuristic mentioned in [31]. It is earlier applied to machine allocation to task. Here the task can be generally considered such as a fragment of codes, a process or even a job executed on computer machine. The idea behind this algorithm is based on that better mappings can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that particular machine is not assigned to it. The sufferage value is defined as below:

$$r_{ij} = w_{ix} - w_{ij} \tag{5.11}$$

where parameters' details are listed in the Table 5.3.

Notation	Description
r_{ij}	Sufferage value of a task t_i if machine m_j can not be assigned to it.
w_{ij}	Task t_i earliest completion time on some machine m_j. i.e. using m_j is the best completion time for t_i.
w_{ix}	Task t_i second earliest completion time on some machine m_x compared to m_j. i.e. assuming m_x is the second best completion time for t_i.

Table 5.3 Sufferage preference algorithm nation

Here is an example to understand this algorithm more clearly. Starting with Table 5.4 which shows the expected execution time values for four tasks on four machines (all initially idle),

	m_0	m_1	m_2	m_3
t_0	40	48	134	50
t_1	50	82	88	89
t_2	55	68	94	93
t_3	52	60	78	108

Table 5.4 Expected execution time matrix

	m_0	m_1	m_2	m_3
t_0	8	2	0	84
t_1	32	6	1	0
t_2	13	25	0	1
t_3	8	18	30	0

Table 5.5 Converted by sufferage preference algorithm

	m_0	m_1	m_2	m_3
t_0	0.25	0.08	0	1
t_1	1	0.24	0.033	0
t_2	0.41	1	0	0.011
t_3	0.25	0.72	1	0

Table 5.6 sufferage preference values

Task	Mapped to
t_0	m_3
t_1	m_0
t_2	m_1
t_3	m_2

Table 5.7 Machine mapping decision

sufferage preference algorithm converts Table 5.4 to Table 5.5 using the Equation 5.11, in which each column has the sufferage values for each task on each machine. For example, compared to the execution time 40 for t_0 on m_0, 48 is the second earliest execution time on m_1 for task t_0. So sufferage value is equal to $48 - 40 = 8$ for t_0. i.e., if m_0 can not be assigned to t_0, task t_0 would suffer 8 execution time units longer. And in the first column, 32 is the biggest sufferage value for t_1, so sufferage preference algorithm decides that m_0 should be mapped to task t_1. In order to make mapping decision easier, the values in Table 5.5 are normalized by the biggest value in each column to get *sufferage preference value* (P) in Table 5.6. For example, based on the machine m_0 the value p for task t_1 is $32/32 = 1$, compared to the value $8/32 = 0.25$ for task t_0. Obviously, the task with the biggest p can get the machine. The same logical way to be applied to generate other columns and in the end the machines mapping decision is achieved in Table 5.7 based on the p values in Table 5.6.

The sufferage preference heuristic considers the "loss" in completion time of a task if it is not assigned to its first choice in making the mapping decisions. By assigning their first choice machines to the tasks that have the highest sufferage values among all contending tasks, the sufferage heuristic reduces the overall completion time. Though the sufferage preference algorithm can not make sure each allocation for the task to get the fastest machine from the machine pool, it can guarantee that all the tasks can be executed in the shortest time period. It considers the whole system instead of the individual task.

5.3.2 Deficiency Preference Algorithm

If the sufferage heuristic is applied to allocate resource from the PCU resource pool to AP, the allocation decision would only consider from the AP point-of-view. But for the entire PCU network system, the allocated server resource utilization has to be another important aspect to be taken into account. Motivated by this consideration, the deficiency preference algorithm is introduced in this section.

The deficiency preference algorithm tries to allocate resource to AP with optimizing that the allocated server resource can server the AP demand exactly. i.e., from the server point-of-view the perfect allocation is that server is allocated to AP when the demand is equal to the server's capacity. If the server's capacity is very close to the AP demand , then this server is highly expected to be assigned to this AP since this allocation is the most efficient use of resource consumption on server side and the most satisfactory meet for AP demand as well. If the server's capacity is much more than AP demand, the preference to assign this server to AP is much lower, because this would waste some spare resources on the server side. If the server's capacity is much smaller than AP demand, the preference is also much lower because this server is not able to satisfy the AP demand completely. In the extreme case, if the server's capacity is infinity and the AP demand is 0, the preference to assign this server to the AP would be 0 since this assignment wastes sever resource entirely, which is never expected. For the PCU system with single service, the deficiency preference value can be expressed by the equation as

$$t_{ij} = \begin{cases} ad_{ij} & 0 \le d_{ij} \le 1 \\ be^{(1-d_{ij})} & d_{ij} > 1 \end{cases}$$

where parameters' details are listed in the Table 5.8.

This formula is also called *demand-provision formula* to express the relationship between demand and provision shown in Figure 5.1, in which 1 stands for the biggest preference value. This allocation is modelled as an optimization problem as well.
The decision variables are defined as follows:

$$u_{ij} = \begin{cases} 1 & \text{if node } j \in S \text{ covers AP group } i \in V \\ 0 & \text{otherwise} \end{cases}$$

Parameter	Notation
k_j	capacity of server$_j$ that denotes the maximum number of concurrent sessions it can handle
h_i	demand at AP$_i$
d_{ij}	$d_{ij} = k_j / h_i$
t_{ij}	the deficiency preference value of allocating server$_j$ to AP$_i$

Table 5.8 Deficiency sufferage preference algorithm notation

With these parameters, the optimization problem for creating a VC becomes (with $i \in V$ and $j,k \in S$):

$$\text{maximize} \quad \sum_i \sum_j t_{ij} u_{ij} \tag{5.12}$$

Subject to

$$\sum_j u_{ij} \geq 1 \quad \forall i \tag{5.13}$$

$$u_{ij} = 0,1 \tag{5.14}$$

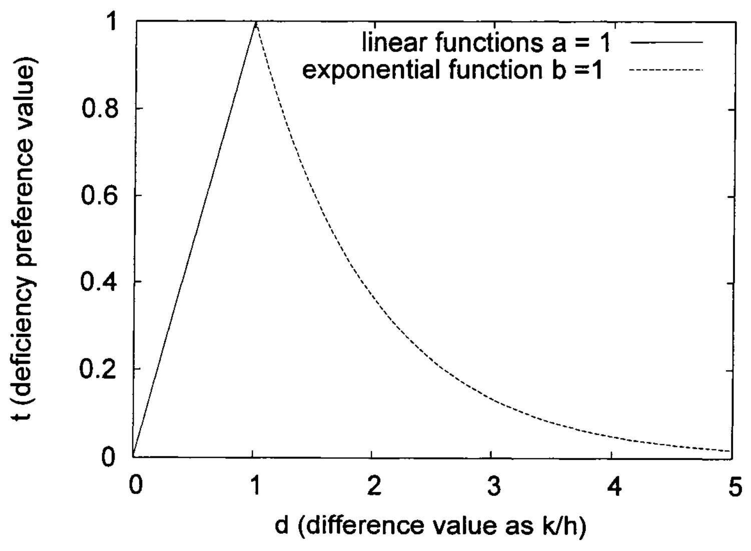

Fig. 5.1 Demand-provision formula graph, with $a = 1$ and $b = 1$

5.3.3 The Optimized Mapping Algorithm

The OMA is run by two parts: one is sufferage preference algorithm, and the other is deficiency preference algorithm. The OMA objective is to find out AP group with the maximized preference value generated from both algorithms to be assigned a server.

Here the sufferage preference heuristic is based on the idea that better mappings can be generated by assigning a server to the VC that would "suffer" most in terms of expected cost if that particular server was not assigned to that VC. The expected cost c_{ij} is considered as the covering cost between the AP group i and server$_j$, which creates the matrix C. Later the deficiency preference algorithm presents the demand-provision relationship as d_{ij} shown in Table 5.9. And the deficiency preference value t_{ij} is the function of d_{ij} shown in Figure 5.1. The algorithm still runs to realize the goal of Equation 5.12 with some constraints mentioned above.

Element ($1 \leq i \leq V$ and $1 \leq j \leq S$)	Description
n_i	the number of nodes in AP group i
a_{ij}	c_{ij} / n_i
k_j	the server$_j$ capacity that denotes the maximum number of concurrent sessions it can handle
h_i	demand at AP group i
r_{ij} (*sufferage value*)	$r_{ij} = a_{ix}$ - a_{ij} (x is the server position in the row i that gives the AP group i the second smallest value than a_{ij})
p_{ij}(*sufferage preference value*)	r_{ij} over the biggest value in the column$_j$ $p_{ij} = r_{ij}$ / Max(r_{ij})
d_{ij}	$d_{ij} = k_j / h_i$
q_{ij}	if server$_j$ is assigned to VC$_i$, $q_{ij} = 1$. Otherwise, $q_{ij} = 0$
g_{ij}	$g_{ij} = t_{ij} + p_{ij}$

Table 5.9 Matrix element

The OMA flowchart in Figure 5.2 illustrates how my heuristic works by step and that two processed phases are indicated as well. In the first phase, because each AP group is supposed to contact one VC for demanding service, the number of VCs is equal to the number of AP groups. In order to solve this allocation problem, a $V \times S$ matrix is composed where V is the number of VCs and S is the number of available resources.

Starting with the covering cost matrix C, the matrix A is computed by averaging the matrix C with c_{ij} over the number of APs in each AP group. This way can help remove the possibility

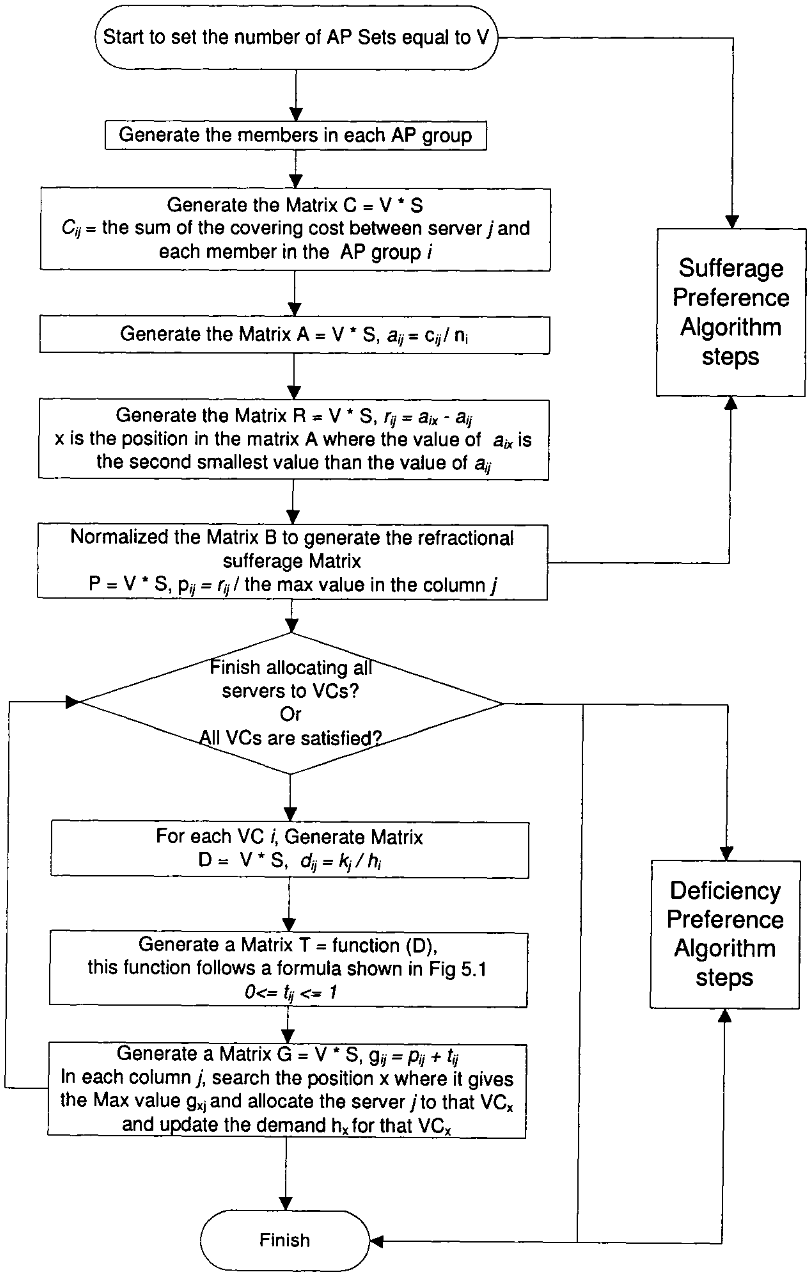
Start to set the number of AP Sets equal to V
Generate the members in each AP group
Generate the Matrix C = V * S
C_{ij} = the sum of the covering cost between server j and each member in the AP group i
Generate the Matrix A = V * S, $a_{ij} = c_{ij} / n_i$
Generate the Matrix R = V * S, $r_{ij} = a_{ix} - a_{ij}$
x is the position in the matrix A where the value of a_{ix} is the second smallest value than the value of a_{ij}
Normalized the Matrix B to generate the refractional sufferage Matrix
P = V * S, $p_{ij} = r_{ij}$ / the max value in the column j
Sufferage Preference Algorithm steps
Finish allocating all servers to VCs? Or All VCs are satisfied?
For each VC i, Generate Matrix D = V * S, $d_{ij} = k_j / h_i$
Generate a Matrix T = function (D), this function follows a formula shown in Fig 5.1
$0 <= t_{ij} <= 1$
Generate a Matrix G = V * S, $g_{ij} = p_{ij} + t_{ij}$
In each column j, search the position x where it gives the Max value g_{xj} and allocate the server j to that VC_x and update the demand h_x for that VC_x
Deficiency Preference Algorithm steps
Finish

Fig. 5.2 The flowchart of resource provisioning algorithms

that higher number or lower number of APs in some given AP group can have more preference to get resource than other AP groups in sufferage preference algorithm and try to make the final allocation be fair for each AP group. After running the sufferage preference algorithm on the matrix A, the first phase ends up with the matrix R stated in the first for loop in Figure 5.3, which can give enough information on which server should be mapped to what AP group in terms of sufferage covering cost. But this first allocation that uses this metric sufferage value only considers the covering cost from the AP perspective in terms of AP group satisfaction and minimizing total covering cost, and never considers the overall resource utilization on the server side. To make the final allocation more practical and efficient, the second step considers the deficiency algorithm in Section 5.3.2 to adjust the preallocation. Here the demand-provision relationship is represented as $d_{ij} = \left(\frac{k_j}{h_i}\right)$. And t_{ij} is the preference value of server$_j$ allocation to VC$_i$ which follows the trend with d_{ij} shown in Figure 5.1, where 1 stands as the top value for deficiency preference.

If d_{ij} is smaller than 1, t_{ij} and d_{ij} both increase. But if d_{ij} is greater than 1, t_{ij} decrease exponentially. i.e., if the server capacity k_j is very close to the VC's remaining demand h_i, then this server$_j$ should be assigned to VC$_i$ since this allocation is the most efficient use of resource consumption on both sides. If the server capacity k_j is larger than VC remaining demand h_i, the preference to assign this server$_j$ to VC$_i$ is lower, as this would waste some spare resources on the server side. If the server capacity k_j is smaller than VC remaining demand h_i, the preference is also lower since this server$_j$ is not a best choice for this VC$_i$ as this wouldn't satisfy the VC completely. In the extreme case, if k_j is infinity and h_i is 0, d_{ij} is infinity so t_{ij} should be almost 0. This formula can be customized by modifying the coefficient value of a or b in terms of the requirement to make the final allocation decision more flexible.

Fitting with the second phase, the matrix R created in the first phase is normalized by being divided by the biggest value in each column to matrix P. Then both p_{ij} and t_{ij} are added up to generate the matrix G. The final server allocation decision is made from the resultant matrix G, in which for each column the position with the greatest value is found and allocate the server to that VC located at that position. The selection process is in a loop until all the servers are selected or each AP group's demand is satisfied with the allocation decision. The pseudocode for the heuristic is shown in Algorithm 5.3. Of course, with the purpose of customization, more weight can be put on p_{ij} or t_{ij} to change the value of g_{ij} in order to modify the final mapping decision.

This OMA take into consideration three main points:

```
Start with the covering cost matrix C and then generate matrix A as $a_{ij} = c_{ij} / n_i$
for each $j$ ( $1 \leq j \leq S$ and in a fixed arbitrary order ) do
  for each $i$ ($1 \leq i \leq V$ ) do
    compute $r_{ij} = a_{ix} - a_{ij}$
    (where $x$ is the position in the row $i$ that gives the second smallest value than $a_{ij}$)
    $p_{ij} = r_{ij} / \max(r_{ij})$
  end for
end for
Initially assign flag $q_{ij} = 0$ ($\forall i \in V, \forall j \in S$)
repeat
  for each $j$ ( $1 \leq j \leq S$ and in a fixed arbitrary order ) do
    for each $i$ ($1 \leq i \leq V$ ) do
      compute $d_{ij} = k_j / h_i$
      compute $t_{ij} = ad_{ij}$ ($0 \leq d_{ij} \leq 1$) or $t_{ij} = be^{(1-d_{ij})}$($d_{ij} > 1$)
      compute $g_{ij} = t_{ij} + p_{ij}$
      In matrix G, search the position $x$ where it gives the biggest value in column $j$ and
      assign $q_{xj} = 1$ and update $h_x = h_x - k_j$
    end for
  end for
until $h_i \leq 0$ ($\forall i \in V$) or $q_{xj} = 1$ ($\forall j \in S$)
```

Fig. 5.3 Resource allocation pseudocode for multiple services

- It considers the balance between demand and provision during the resource allocation with the metric deficiency preference value to make the resource utilization in the whole system as big as possible. At the meanwhile, make the entire system cost minimized.
- It can be customized by putting different weight on both preference values, sufferage preference value and deficiency preference value, to change the resultant matrix value based on the different SLA requirement from different AP group.
- For the large-scale network, it is much simpler and faster to run compared to the algorithm presented in [9]. OMA running simulation time is only one tenth of the one spent for the algorithm in [9] based on the same number of nodes in the network.

5.4 An Example of the Creation for Multiple VCs

Here a simple example of the creation process is illustrated for multiple VCs on a 15-node network. The interconnection topologies among the resources were generated using an Internet topology generator called the Tiers [32]. The Tiers creates a network with WAN-MAN-LAN topology. The network initialization code written in Matlab accepts this network information file and places the 11 APs at randomly chosen as LAN nodes and assume the rest of 4 nodes to be possible candidates for the location of serving resources. It also assigns random demands for the APs and random capacities and capabilities for the servers and then generates the covering cost matrix between each AP and server for the network. The VC randomly selected AP to be served as shown in Table 5.10.

The output of this initialization is a text file that is processed to be Table 5.11, of which each element is the sum of the covering cost. For example, there are 5 APs connecting to VC_1, so c_{11} represents the sum of the covering cost between each AP in VC_1 and S_1. In order to ignore the effect of the number of APs in each VC and treat each VC fairly in terms of covering cost for the future resource allocation, Table 5.11 is transformed to be Table 5.12 for the average cost of each VC. Therefore, none of the VCs can have higher priority to get resources in terms of the bigger number of AP members inside. In this simulation, VC_1 has 5 AP members inside as shown in Table 5.10, so $a_{11} = c_{11} / 5$, with the same way to compute the other elements in Table 5.12 (starting with Table 5.12, the multiple VCs creation code, written in Matlab, contains codes for implementing OMA heuristic).

	Number of APs
VC_1	5
VC_2	3
VC_3	3

Table 5.10 AP Random Selection

	S_1	S_2	S_3	S_4
VC_1	1600	1920	3600	2000
VC_2	1200	1968	2112	2136
VC_3	1320	1632	2256	2232

Table 5.11 Matrix C for the covering cost between AP and server

	S_1	S_2	S_3	S_4
VC_1	320	384	720	400
VC_2	400	656	704	712
VC_3	440	544	752	744

Table 5.12 Matrix A for the average covering cost of each VC

The creation of multiple VCs is carried out in two stages: in stage one, the VC creation code is run with the sufferage preference algorithm, which generates Table 5.13. With the way mentioned in the pseudo-code shown in Figure 5.3, if VC_1 is assigned S_1, it will result in the best covering cost for V_1. But if S_2, which is the second best, is assigned to VC_1, VC_1 would suffer more 64 in terms of covering cost than by being assigned S_1. Therefore, the sufferage element b_{11} = a_{12} - a_{11}. After every element r_{ij} is generated, for the server$_j$, we can decide which VC would get most sufferage if that server$_j$ isn't assigned to this VC_i. For example, S_1 should be assigned to VC_2 since b_{21} has the greatest value (256) in Table 5.13. In order to make it consistent with the next stage, Table 5.13 is normalized to Table 5.14 by column.

In the second stage, considering the difference between VC demand and server provision, Table 5.15 is created. In Table 5.15, the element is computed based on the formula in Equation 5.12. Here, the coefficients a and b are equal to 1. In this example, p_1 (1519) is smaller than r_1(2917), so the difference $\frac{p_1}{r_1}$ is smaller than 1. Therefore, the first formula is chosen to get the t_{11}. Element t_{22}, however, is created in the second way of this formula. Here p_2 (2456) is bigger than r_2(725), so their difference is bigger than 1. $t_{22} = e^{1-d_{22}}$, where d_{22} = p_2 /r_2. To consider both effects of matrix T and H on the allocation decision, Table 5.14 and Table 5.15 are added by corresponding matrix element to get Table 5.16. In each column, the server is mapped to the VC which gives the biggest number. From these results, the initial mapping decision is shown in Table 5.17. This example shows that all the servers are allocated to VCs in the first loop. It already meets one of the conditions mentioned in my algorithm so the allocation ends up with this mapping decision.

	S_1	S_2	S_3	S_4
VC_1	64	16	0	320
VC_2	256	48	8	0
VC_3	104	200	0	8

Table 5.13 Matrix B generated by sufferage preference algorithm

	S_1	S_2	S_3	S_4
VC_1	0.25	0.08	0	1
VC_2	1	0.24	1	0
VC_3	0.40625	1	0	0.025

Table 5.14 Matrix H normalized by Matrix B

	S_1	S_2	S_3	S_4
VC_1	0.5207	0.1395	0.1503	0.1711
VC_2	0.0355	0.03379	0.1302	0.1376
VC_3	0.4068	0.7089	0.1375	0.7871

Table 5.15 Matrix T generated by demand-provision formula

	S_1	S_2	S_3	S_4
VC_1	0.7707	0.2195	0.1503	1.1711
VC_2	1.0355	0.2737	1.1302	0.1376
VC_3	0.81305	1.7089	0.1375	0.8121

Table 5.16 Matrix F, the sum of Matrix T and Matrix H

Server	Mapped to
S_1	VC_2
S_2	VC_3
S_3	VC_2
S_4	VC_1

Table 5.17 Mapping decision

Chapter 6

Experimental Results and Discussions

This chapter explains the simulation study to evaluate the performance of multiple VCs configurations based on the performance model of an online game service. This performance is compared with the one of the creation mechanism for single VC described in [9]. Section 6.1 discusses the assumptions made in performing the simulation. Section 6.2 describes the simulation in detail. The obtained results and detailed discussion are given in Section 6.3.

6.1 Assumptions

For this study, the computation time at the server is assumed to be proportional to the session size and there is sufficient bandwidth assumed between the player and the game server to meet the requirement.

For the session attributes, Chang and Feng ([18]) already proved that the *probability density function* (PDF) of player session duration can be fitted accurately with a Weibull distribution as Equation 6.1, which has three parameters β, η and γ. In this form, β is a shape parameter or slope of the distribution, η is a scale parameter, and γ is a location parameter. As the location of the distribution is at the origin, γ is set to zero, giving the two-parameter form for the Weibull PDF as Equation 6.2. They use a probability plotting method [33] to estimate the shape (β) and scale (η) parameters of the PDF of the session time. A Weibull distribution with $\beta = 0.5$, $\eta = 20$, and $\gamma = 0$ closely fits the PDF of the measured session times for the one-week trace of a popular first person shooter online game – Counter-Strike.

$$f(T) = \left(\frac{\beta}{\eta}(\frac{T-\gamma}{\eta})^{\beta-1}e^{-(\frac{T-\gamma}{\eta})^{\beta}}\right) \tag{6.1}$$

$$f(T) = \left(\frac{\beta}{\eta}(\frac{T}{\eta})^{\beta-1}e^{-(\frac{T}{\eta})^{\beta}}\right) \tag{6.2}$$

The session length in my simulation is assumed to follow this Weibull distribution with the same parameters ($\beta = 0.5$, $\eta = 20$, and $\gamma = 0$). Sessions are assumed to arrive at the VCs with Poisson distributed inter-arrival time. It is assumed that for different APs, the mean session intervals vary in the range of 1 to 10 seconds and the mean session lengths vary in the range of 100 to 720 seconds. The average demand from each AP can be generated by the mean session length divided by the mean session inter-arrival.

The demands at the APs are assumed to be steady over the run time of the simulation. It may be not totally realistic but acceptable for the purpose of this study. The purpose of this simulation study is not to address such complicated situations but to evaluate the fundamental performance of the proposed system.

6.2 Simulation Setup

The simulation study is carried out in three phases. The first phase is to create the large scale network with Tiers [32], which has a total of 314 nodes, 179 of them being APs. The Tiers-created network is a WAN-MAN-LAN topology illustrating the node locations, connectivity details, and the delay and bandwidth of the links.

In the second phase, the network initialization code written in Matlab accepts this network information file, places the APs at randomly chosen LAN nodes and distributes them into different AP groups. The rest of the nodes are candidates for the servers and assigned random capacities and capabilities. And then the code generates the cost matrix c_{ij} for the network. Later the centralized *optimized mapping algorithm* (OMA) heuristic is run to create the multiple VCs. The created VC has the information on what servers are mapped to serve which AP group. The example of the phase implementation is illustrated in the Chapter 5.4.

The final phase creates the working network model, generates the load and evaluates the per-

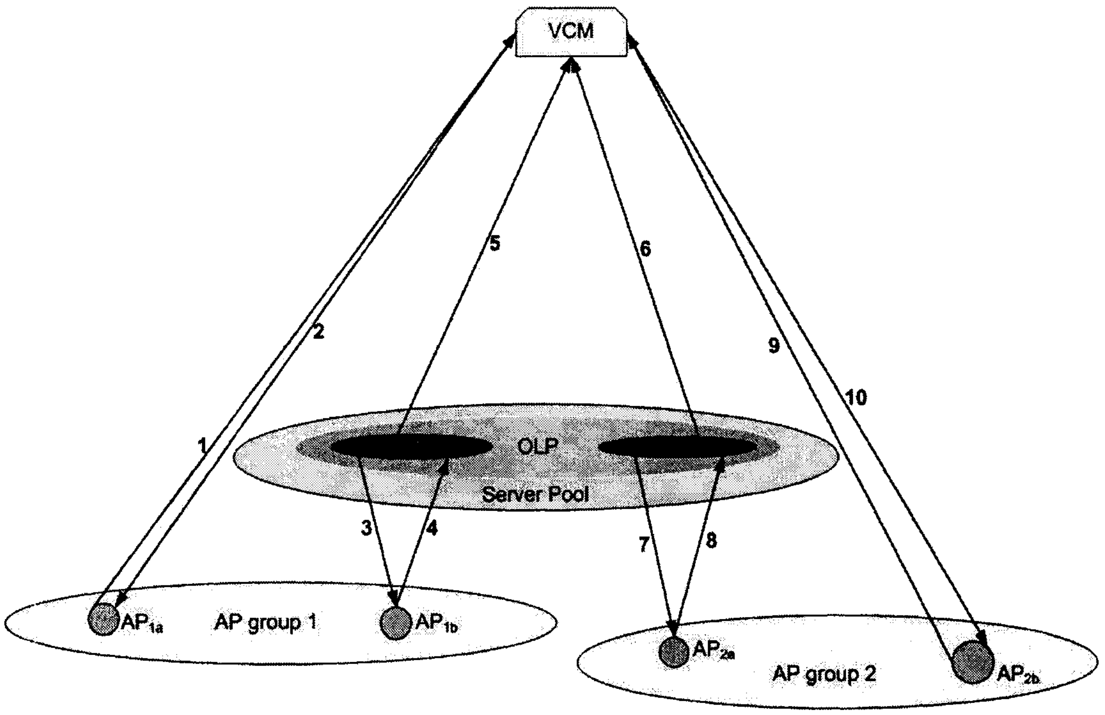

Fig. 6.1 Creation of multiple VCs in PCU resource management system

formance of the created VC with a discrete event simulator called Parsec [34]. Figure 6.1 shows the main components in the simulation setup: AP, VC and *virtual cluster manager* (VCM) established in a server node that is not participating in any VC. However, in real cases, the VCM can be distributed in each VC. The VCM is dedicated to coordinate the scheduling of sessions. When the VCM receives the request from AP, it uses the output of the performance model described in Chapter 4 to make a decision. This performance model has the following primitives as input: (i) the number of current server and client bots on that selected game server (ii) the current map (iii) two corresponding values for the given map: one is the C_M^m and the other is C_M^n. In this simulation, the output of the performance model is expressed in the percentage of the used capacity on a given server with these input information. Because the candidate server has not yet served this session, the predicted result is based on the condition that this session is already served on that server. If the result is bigger than one, the VCM will send rejection to the AP and AP will wait until VCM notices this AP group of free capacity in this VC to server this session. Otherwise, the VCM will send acceptance decision to the AP. The Parsec codes consist of entities which are the templates of different types of nodes in the network: AP, server, and VCM. And it can provide the functionalities of the individual type, such as session generation, communication, session processing. The entities communicate through application level messaging.

The simulation process starts with AP_{1a} sending the request to the VCM (process 1). VCM processes this request to map the session to one server, and then informs the AP_{1a} (process 2) with one decisions. After receiving the successful mapping decision, AP_{1b} will communicate with the assigned server in VC_1 (process 3 and 4). At the same time, VC talks to the VCM periodically about its status (process 5 and 6) such as the number of the current bots, the number of real players and map type. Because the second phase has already made the decision of multiple VCs allocation to serve the AP group, the VCM knows which VC can serve the session from the given AP group, i.e., the request 1 will be scheduled on VC_1, and the request 9 will be scheduled on VC_2.

Generally the maps in the game world are switched periodically based on the game service configuration. This scenario can be simulated by switching maps at scheduled times. The simulation time is divided into three identical slots. When the game proceeds, the number of current bots varies randomly in the range of 1 to 12 to increase the game competition interest or prevent the boredom with the lack of real players on the game server. Depending on the conditions present in the SLA, the multiple VCs creation can be triggered periodically to make different decision on scheduling the demand from the AP group.

There is an optional component – *overload partition* (OLP) in the working network model shown in Figure 6.1. It is used to handle the variations in demand from the expected value. The OLP can be considered as a protection layer of the actual VC. For each VC, OLP is created in two steps. At first, a resource partition is created for the service considering an increase in demand condition. And then the VC nodes are extracted from the resource partition obtained from the first step by rerunning the allocation process with nominal demand condition. The resources outside the VC set, but inside the resource collection from the first step, are allocated as the OLP nodes. When the VC is created, the resources are dedicated for the exclusive use of the service hosted by the VC, while the resources in the overload partition are shared among VCs to handle unexpected spikes in demand from APs. If VCM can not select one available resource to serve the upcoming session in the specific VC, it will try to map one suitable server in the OLP to handle this session.

Because a VC is a large-scale system with a wide geographical presence, it is unrealistic to assume the demand is uniform across the VCs. The simulation study analyzes the performance of multiple VCs under different load conditions: nominal load, underload and overload. The following styles of load variations are considered in this simulation:

- *Fixed loading:* Demands from all APs in each AP group are increased by the same factor.
- *Window-framed loading:* The demand change factors for AP groups are randomly selected from a "window", so that some of them are above nominal values and others are below. The center of the window is shifted moving the overall loading pattern of the system towards underloaded or overloaded condition.
- *Asymmetrical loading:* The demands in a small percentage of the AP groups are increased by a particular factor and the demands in the rest are reduced so that the whole system is at the nominal load conditions.

These different loading schemes are used to represent different practical loading scenarios. Demands of the APs are varied either by changing the session intervals or the session lengths.

6.3 Results and Discussions

6.3.1 Performance Metrics

In our experimental setup, there are three main performance metrics to determine whether multiple VCs configuration by OMA can make the whole system work efficiently and fairly under the applied workload. One of these metrics is collected at the server side, one is at the client side and another one is for both sides considering cost c_{ij} between them:

- *Client side metric:* A performance metric *satisfaction factor* (SF) shows how much percentage of sessions served satisfied in the AP group. Before computing the metric SF, a *threshold cost* (TC) should be defined firstly. TC helps to decide whether this session can be served with content or not. If the session is served with cost above TC, the session is served unhappily. Otherwise, it is served with content. SF is defined as the number of sessions served under TC divided by the whole number of sessions.
- *Server side metric:* A *resource utilization percentage* (RUP) is another performance metric introduced to consider the server part in VC system, which is computed by the used capacity in a VC over the VC's entire capacity.
- *Client-server metric:* The system is also tested with another metric called *unit utilization cost* (UUC), Let θ be the number of servers acquired at any given time, τ be the fixed

cost of using the server per unit time, and λ be the overall resource utilization. With these parameters, the unit utilization cost is defined as $\theta \times \tau / \lambda$. A lower value for this metric indicates a cost-effective configuration.

6.3.2 Simulation and Discussion

Generated from the large scale 314-node network, the multiple VCs created for AP groups are shown in Table 6.1. The simulations are carried out for a fixed duration of time.

	VC nodes	AP Group nodes	AP Group demand
VC_1	5	19	1512
VC_2	11	33	3163
VC_3	10	36	3049
VC_4	6	23	1592
VC_5	6	13	1226
VC_6	15	55	4413
Total	53	179	14955

Table 6.1 Server allocation in each VC

TC is defined considering the SLA of each AP group and the network cost condition. If the TC is set to a big number, the session can be served with content easily. Otherwise, it is hard to server the session under this TC. For the extreme example, the cost between every AP and the allocated VC ranges from 100 to 400. If the TC is set to 410, all the sessions would served with content and the SF is 100%. Otherwise, if we set TC to 50 for this network, the SF will be 0. In my simulation 500 is chosen as a testing TC based on the cost range from 147 to 4280 in my network model. The more accurate way to compute TC for each AP group will be done in future investigation. Particular attention is paid to the set of sessions that are served under the threshold cost. All the values presented in these plots are averages of at least five runs.

Figure 6.2 shows the performance with varying normalized mean session intervals. The normalized session interval represents the nominal load condition for each VC. The longer the mean session interval, the lower the normalized load and the higher the satisfaction factor can be. The same session length and different intervals are chosen to test the satisfaction factor. The plot shows that for each VC, the satisfaction factor moves smoothly from underloaded to nominal loaded and then goes to overloaded conditions. It also proved that this resource allocation based on the AP group demand can serve the AP without SF spikes which can be brought by the variant loaded condition.

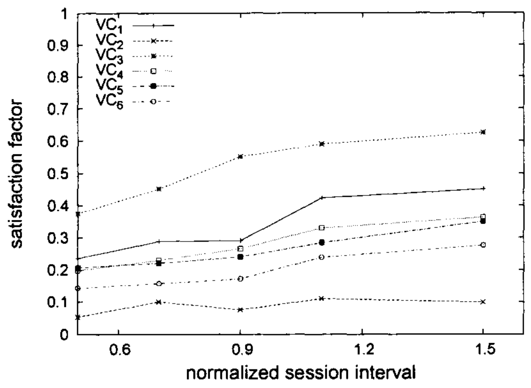

Fig. 6.2 Performance of multiple VCs with mean session interval

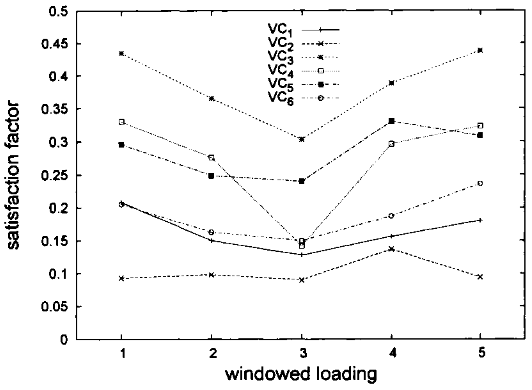

Fig. 6.3 Performance of multiple VCs with window-framed loading

Window-framed loading condition tag	Session interval window size	Session interval window center	Session length window size	Session length window center
1	0	1	0	1
2	0.67	1.33	0	1
3	1.5	1.25	0	1
4	0	1	0.5	1
5	0	1	1.5	1.25

Table 6.2 Window-framed loading details

The VC configurations is simulated under different demand situations to verify whether the VC is able to fairly treat the demand from the nodes with average satisfaction factor. Figure 6.2 shows the performance under window-framed loading. Here the demand varies in a window. The session length and session interval at each AP group are selected from a uniformly distributed window with details shown in the Table 6.2. The first row of the table presents the nominal loading condition. The second and third row keeps session length at the nominal loading and varies session interval. For example, the third row shows demands generated with a window of width 1.5 - centered at 1.25 times nominal loading for session interval. Therefore, the maximum value for this window-framed loading is 2 times the nominal loading and the minimum value is

0.5 times the nominal loading, with session length being the same. The fourth and fifth row varies session length but keeps the session interval constant as nominal loading.

The results shown in the Figure 6.3 indicate that VC_3 can handle this window-framed situation better than the other VCs with higher SF. All the SF lie within 10% and 45% and the SF for each AP group follows the similarly symmetrical trend shaped like V. The bottom point of the shape is positioned at the window-framed loading condition tag 3. Because each AP group can response similarly to the same window-framed loading conditions, it suggest that this allocation algorithm can treat every VC relatively fairly.

Figure 6.4 shows the result in asymmetrical loading conditions. The APs in each group to be overloaded are randomly selected. Table 6.3 indicates the asymmetrical loading details for each conditions tag shown in X axis of Figure 6.4. For example, for the condition tag 2, 25% randomly selected APs are overloaded by the factor 1.1, the rest 75% APs in each AP group are underloaded so that the final load can be at nominal loading conditions. From this plot, the satisfaction factor lies mainly from 10% to 45%. The results are similar: there is no significant difference between each VC, though VC_3 can handle this situation better than the other VCs.

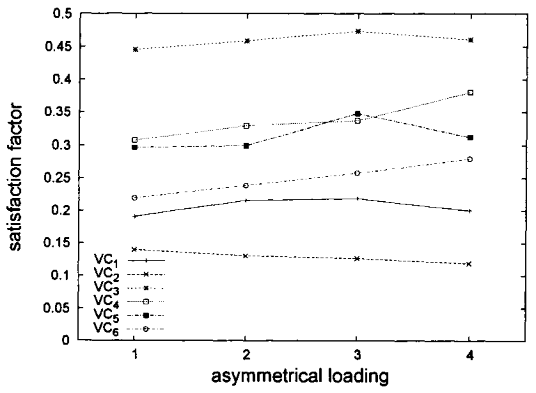

Fig. 6.4 Performance of multiple VCs with asymmetrical loading

Figure 6.5 shows the RUP for each VC with the simulation time proceeding. The horizontal axis shows the scaled simulation time while the vertical axis shows the total used capacity percentage in each VC cluster. Among all the VCs, VC_3 has the most spikes in RUP from 65% to 75% during the whole simulation time. VC_5 and VC_6 have increasing value in RUP from 65% to 80% during the time slot from 20 to 40 but keep stable RUP in other time periods. Other VCs have RUP with less variation than VC_3 from 72% to 80%. Even though the RUP follows different

Asymmetrical loading condition tag	Normalized overload factor	% of overloaded APs
1	1	1
2	1.1	0.25
3	1.2	0.25
4	1.5	0.25

Table 6.3 Asymmetrical loading details

trend for each VC in this simulation, the whole VC system can keep RUP mainly from 65% to 80% , which helps to suggest that OMA can give each VC efficient resource utilization from the point of server side view.

Figure 6.6 shows that most VCs configurations hold UUC metric steadily and VC_2 keeps the UUC with the least oscillations. The smallest UUC for VC_1 means that VC_1 works in the most cost-effective way when serving the AP group. Even though OMA can not keep every VC to work in a cost-efficient way, each VC can work with the resources allocated by OMA in a similar way that UUC keeps little oscillated.

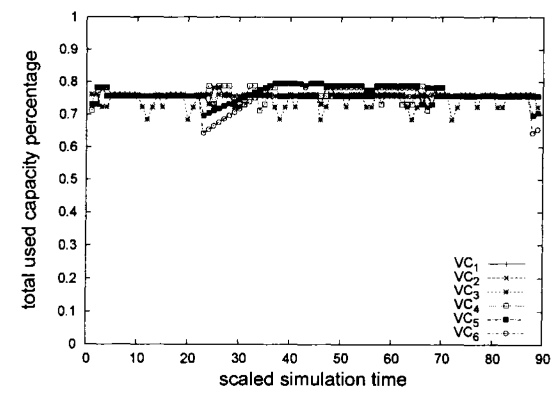

Fig. 6.5 Resource utilization within each VC

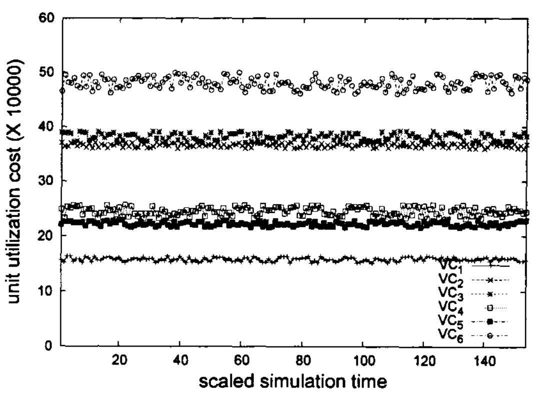

Fig. 6.6 Unit utilization cost of each VC

Furthermore, my resource allocation algorithm (*OMA*) for multiple services is compared with the one (*SSA*) studied in [9] for single service. The comparison is made from the perspective of two different scenarios (with OLP and without OLP) in terms of RUP. The average RUP computed out of all 6 VCs by OMA is compared to the RUP obtained from one single VC by SSA. Figure 6.7 shows that with OLP the OMA outperforms SSA although by a smaller margin of RUP, but the RUP by OMA is not as stable as the one by SSA because the RUP range from 53% to 57% by SSA is covered inside the RUP range from 52% to 60% by OMA. On the contrary, SSA outperforms OMA without OLP condition by 20% in terms of RUP.

The comparison is also made on RUP under two different scenarios from the same algorithm. It is shown apparently that the RUP is reduced with OLP condition from without OLP condition for both OMA and SSA algorithms. This result is expected because the OLP resources are shared with multiple VCs and they by design should have low utilizations when observed from an individual VC. The OLP resources can expect loading from VC when overloading comes to it.

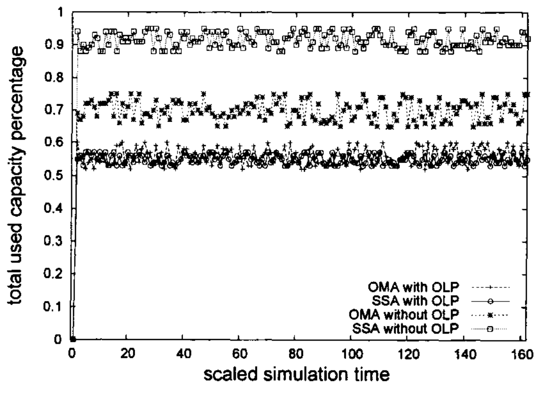

Fig. 6.7 Algorithm comparison runs with and without OLP

Overall, OMA is relatively insensitive to OLP (the average difference in RUP is 13% with or without OLP) while SSA is quite sensitive to OLP (the average difference in RUP is 48% with or without OLP). OMA can make the VC system more independent on OLP under different loading conditions.

Based on the above simulated experimentations, it can be concluded that OMA can handle fairly for multiple VCs from both AP group side in terms of SF and server side in terms of RUP under different loading conditions. But there is still much space to improve this algorithm because OMA fails to keep every VC in high value in terms of UUC.

Chapter 7

Conclusions and Future Work

7.1 Contributions

This thesis makes the following contributions to online games hosting on wide-area computing utilities:

- *Utility-aware performance model for online game:* In this work, a set of basic benchmarks is introduced to measure the basic capacities of a game system and to reveal the main CPU bottleneck. They allow one to derive the scaling rules of server capacity. Using experimental testbed, a utility-aware performance model for the game server "Twiligh" is designed. It uses the basic benchmark measurements to estimate resource requirements for a particular session, in order to compute the server capacity for delivering a realistic quality of service.

- *Adaptive resource management system design:* This work integrates the game performance model with a PCU on-demand infrastructure, which gives a convenient, flexible system to dynamically support games. Generally, clusters are designed for maximum demand conditions resulting in poor overall resource utilization. Here, an alternative approach (MBRP merged approach) is presented with VC to host online games. Simulation results shows that MBRP is an effective way to deal with complex resource management challenges.

- *Resource allocation algorithm for multiple VCs :* This algorithm deals with how to allocate resources for multiple services simultaneously in PCU environment. Simulations show that this resource allocation can treat each VC fairly based on the demand generated by each AP group.

7.2 Limitation and Future Work

Limitations listed below provide opportunities to my work.

- *Mixed Service Types.* The service type considered in my simulation study is the online FPS game. Other types of popular games such as MMORPG can be merged into this system but the performance models needs further modifications to be more accurate to match their special features because game design and player behavior exert a significant impact on the resource usage model at the server end. The heuristic code for multiple services resource allocation needs no modifications.
- *More Realistic Testbed.* In this study LAN environment and simulated trace is used to develop the performance model; a more realistic estimate would consider more resource consumption and constraint in WAN environment with real trace.
- *Distributed Algorithms.* The distributed version of the proposed sufferage preference and deficiency preference algorithm can be developed to make the allocation process on-line.

Appendix A

Abbreviations and Acronyms

AP	Anchor Point
FLP	Facility Location Problem
PCU	Public Computing Utility
QoS	Quality of Service
SLA	Service Level Agreement
VC	Virtual Cluster
C/S	Client Side
S/S	Server Side
CPU	Center Processor Unit
FPS	First Person Shooter
TIO	Tivoli Intelligent Orchestrator
SNMP	Simple Network Management Protocol
MMORPG	Massively Multiplayer Online Role Playing Game
PDF	Probability Density Function
SSA	Single Service Algorithm
LAN	Local Area Network
WAN	Wide Area Network
APM	Autonomic Program Manager
CAs	Computer Agents
VPP	Variable Problem Partition
MMOG	Massively Multiplayer Online Game
SPV	Sufferage Preference Value
SF	Satisfaction Factor
TC	Threshold Cost
RUP	Resource Utilization Percentage
UUC	Unit Utilization Cost

References

[1] "AOL Corporation America Online Press Data Points." http://corp.aol.com/press/press-datapoints.html.

[2] "An analysis of mmog subscription growthcversion 21.0, mmogchart.com online publication." http://www.mmogchart.com/.

[3] "IBM Corporation The On Demand Operating Environment." http://www.ibm.com/ebusiness/ondemand/us/overview/operating-environment.shtml.

[4] "On Demand Business IBM Corporation." http://www.ibm.com/ondemand.

[5] "HP Corporation Utility Data Center: Solutions." http://www.hp.com/solutions1/infrastructure/solutions/utilitydata/index.html.

[6] "HP Corporation Utility Data Center: HP's First Proof Point for Service-Centric Computing. IDC white paper.." http://www.idc.com.

[7] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat, "Model-based resource provisioning in a web service utility," in *Fourth USENIX Symposium on Internet Technologies and Systems*, pp. 57–71, January 2003.

[8] L. Cherkasova and L. Staley, "Building a performance model of streaming media applications in utility data center environment," in *Proceedings of ACM/IEEE Conference on Cluster Computing and the Grid (CCGrid)*, pp. 36–43, May 2003.

[9] B. Maniymaran and M. Maheswaran, "Virtual clusters: A dynamic resource coallocation strategy for computing utilities," in *16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, pp. 166–168, November 2004.

[10] M. Welsh, D. E. Culler, and E. A. Brewer, "SEDA: An Architecture for Well-Conditioned, Scalable Internet Services," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pp. 230–243, April 2001.

[11] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, and et al., "Océano – SLA based management of a computing utility," in *IEEE/IFIP International Symposium on Integrated Network Management*, pp. 24–26, May 2001.

[12] K. Shen, H. Tang, T. Yang, and L. Chu, "Integrated resource management for cluster-based internet services," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 225–238, 2002.

[13] A. Abdelkhalek, A. Bilas, and A. Moshovos, "Behavior and performance of interactive multi-player game servers," in *In the Proc. International IEEE Symposium on the Performance Analysis of Systems and Software*, November 2001.

[14] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: evidence and implications," in *INFOCOM*, pp. 126–134, March 1999.

[15] A. Wolman, M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the scale and performance of cooperative web proxy caching," in *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pp. 16–31, December 1999.

[16] F. Chang, W. Feng, W. Feng, and J. Walpole, "Provisioning on-line games: a traffic analysis of a busy counter-strike server," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 3, pp. 18–18, 2002.

[17] T. Henderson and S. Bhatti, "Modelling user behaviour in networked games," in *MULTIMEDIA '01: Proceedings of the ninth ACM international conference on Multimedia*, pp. 212–220, May 2001.

[18] F. Chang and W. Feng, "Modeling player session times of on-line games," in *NetGames '03: Proceedings of the 2nd Workshop on Network and System Support for Games*, pp. 23–26, June 2003.

[19] D. Menascè and V. Almeida, *Capacity Planning for Web Performance: metrics, models, and methods.* Prentice Hall PTR, 1998.

[20] M. Ye and L. Cheng, "System-performance modeling for massively multiplayer online role-playing games," *IBM System Journal.*, vol. 45, no. 1, pp. 45–49, 2006.

[21] A. Shaikh, S. Sahu, M. C. Rosu, M. Shea, and D. Saha, "On demand platform for online games," *IBM System Journal.*, vol. 45, no. 1, pp. 7–11, 2006.

[22] B. Lee and J. Weissman, "Dynamic replica management in the Service Grid," in *IEEE 2nd International Workshop on Grid Computing*, pp. 433–434, November 2001.

[23] G. Deen, M. Hammer, J. Bethencourt, I. Eiron, J. Thomas, and J. H. Kaufman, "Running quake II on a grid," *IBM System Journal.*, vol. 45, no. 1, pp. 21–25, 2006.

[24] D. Saha, S. Sahu, and A. Shaikh, "A service platform for on-line games," in *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, pp. 180–184, September 2003.

[25] "Quakebot c/s." www.unconventional-wisdom.org/QuakeBot/quakebot.htm.

[26] "Quake-c tutorial." http://www.inside3d.com/qcspecs/qc-menu.htm.

[27] "Frikbot tutorial." http://www.inside3d.com/frikbot/projects.shtml.

[28] "Mikebot tutorial." http://www.planetquake.com/mikebot/table-index.html.

[29] M. Daskin, *Network and Discrete Location: Models, Algorithms, and Applications.* John Wiley & Sons, Inc., 1995.

[30] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Analysis of a local search heuristic for facility location problems," in *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1–10, January 1998.

[31] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–131, 1999.

[32] M. B. Doar, "A better model for generating test networks," in *IEEE Globecom*, pp. 86–93, November 1996.

[33] R. Corporation, "Life Data Analysis and Reliability Engineering Theory and Principles Reference from Reliasoft," 2003. http://www.weibull.com/lifedatawebcontents.htm.

[34] "Parsec: Parallel simulation enviornment for complex systems." http://pcl.cs.ucla.edu/projects/parsec/.