Advisor: Rua-Huan Tsaih, Professor, MIS, NCCU

Fang Yu, Assistant Professor, MIS, NCCU

iPalace Video Channel: The Service and Its Implementation

故宮影音頻道:服務及實作

by

Tzu-Liang Yang

Department of Management Information Systems

NATIONAL CHENGCHI UNIVERSITY

2012

Abstract

The iPalace Video Channel is a video-based website and provide people around the world a brand new video-broadcasting service on the Internet which introduces relics in National Palace Museum in Taiwan. The iPalace Video Channel is designed to meet the high standard of service quality which users can watch videos without delay and be capable of coping with potential high peak demands from worldwide. In this paper, first, we establish the iPalace Video Channel and an elastic and reliable mechanism via load balancing that is capable of handling potential huge peak-demand of video services. Second, we conduct an experiment by sampling the video-based service in practice and evaluating the presented approach online. Third, from the finding of evaluation, we establish the two phase service system for delivering video services with high standard of service quality. Finally, we discuss the formal queuing model of video services from which the settings of the queuing system that satisfy acceptable service quality could be calculated under various computer and network environments

Keyword: video-based service, cloud computing, load balancing, queuing theory

Index

1. Introduction

Originality and content play a significant role in service economy where people tend to have higher mental needs than material needs for the uprising of quality of life. As a result, in Taiwan, cultural and creative industries are booming prospectively. National Palace Museum in Taipei, Taiwan, with a huge amount of collections of high-quality Chinese artifacts among thousands of years, is asked by the government to build a cultural and creative platform to promote the cooperation and the exchange between the academics and the cultural and creative industries. On the other hand, while the advancement of information and telecommunication technologies (ICT)[1] causes rapidly changing of modern technologies, it makes the ICT-enabled service[2] become a reality and also make people, organizations, systems and heterogeneous devices link more efficiently and cost less than ever before. In response to the ICT advancement and the wave of globalization, NPM has devoted great efforts in recent years to complete the Digital Archives Project and to increase its cultural influence. (National Palace Museum, 2010)

Furthermore, by adopting advanced ICT and new media, the NPM hopes to break through the Museum's physical boundaries, making its treasured artifacts and educational resources reach the young generation and enter the world. Thus, the NPM would like to follow the 4th Phase Taiwan E-Government project (Research Development and Evaluation Commission, 2011) to send service to portable devices to increase the interaction between young people and NPM. From the academic perspective, the NPM service is evolving to be customer-oriented, innovative, and ICT-enabled. That is, to meet the evolving needs of customers (especially the young generation) and to create leverage from these innovations, one of solutions for the NPM is well applying ICT to provide on-line, real-time, interactive services. Through ICT-enabled services, the NPM can conquer limitations of time, space, human resources, and so forth, and improve efficiency and competitiveness.

[1] Currently, the ICT advancement is caused by a convergence of several technologies: Internet, global telephone system, communications standard TCP/IP (Transfer Control Protocol/Internet Protocol), addressing system of URLs, personal computers and cable TV, customer databases, user-friendly free browser, portable devices (tablet PC, smart phone), sound, graphics and video.

[2] The ICT-enabled service is the on-line, real-time, interactive service provided through the ICT application.

In this paper, we introduce the iPalace Video Chanel service and the design and implementation of the iPalace Video Channel - a full video-based website on an elastic cloud computing platform. The iPalace Video Channel introduces NPM and delivers the beauty of historical relics to people by videos. The structure of the iPalace Video Channel is designed for satisfying the high standard service quality and the load balance under the scenario of huge peak-demand for video services. Initially, we design a load balancing mechanism for handling massive requests. After the online experiment, we find that even though we dispatch requests in balance, the peak load which will violate the service quality still exist. Thus, we design a two phase service system for the iPalace Video Channel. The service system regarding the phase I is a M/D/1: ∞/∞/FCFS queuing system which accomplishes load balancing mechanism and satisfies the service quality. The service system regarding the phase II is a M/M/1: ∞/BOUND/round-robin queuing system which defines the constraint and helps us to figure out the best solution for satisfying service quality.

Our objectives are: (1) to establish the iPalace Video Channel on the cloud, (2) to establish an elastic and reliable mechanism via load balancing that is capable of handling potential huge peak-demand of video services, (3) to conduct an experiment by sampling the video-based service in practice and evaluating the presented approach online, (4) according to the result of experiment, to establish the two phase service system for delivering smooth video services, and (5) to investigate the formal queuing model of video services from which the settings of the queuing system that satisfy acceptable service quality could be calculated under various computer and network environments.

We will make literature review in section 2, the iPalace Video Channel service design and implementation discuss in section 3, the Service System of the iPalace Video Channel discusses in section 4, and section 5 discusses online experiment and the result and makes conclusions in section 6.

2. Literature Review

2.1 Service Oriented Architecture

Service science is the study of service systems, which are dynamic value co-creation configurations of resources which include people, technology, organizations, and shared information and aims to create a basis for systematic service innovation. Service systems are considered the basic unit of analysis in service science. The normative function of service systems is to connect people, technology and information through value propositions with the aim of co-creating value for the service systems participating in the exchange of resources within and across systems. Service science would combine organization and human understanding with business and technological understanding to (1) explain the origins and growth of service systems; (2) solve fundamental problems such as how to invest optimally to improve service productivity and quality; and (3) produce unique service professionals and service scientists. (Maglio and Spohrer 2008; Vargo and Akaka 2009)

Service-oriented architecture (SOA) presents an approach for modeling and packaging software as a set of services by building distributed systems that deliver application functionality as services to either end-user applications or other services. SOA is proposed when enterprises face the challenge of Heterogeneity and change. Enterprises contain a lot of different systems to provide services to customers. Integrating products from multiple vendors and across different platforms is a challenge for enterprises. Globalization and e-business are accelerating the pace of change and lead to fierce competition which leads to shortening product cycles. Customer needs and requirements change more quickly driven by competitive offerings and wealth of product information available over the Internet. Business must rapidly adapt to survive and the IT infrastructure plays an important key. In order to alleviate the problems of heterogeneity, interoperability and ever changing requirements, such architecture should provide a platform for building application services with the characteristics of loosely coupled, location transparent and protocol independent. (Endrei et al. 2004; Namjoshi and Gupte 2009)

Based on such a service-oriented architecture, a service consumer does not even have to care about a particular service it is communicating with because the underlying infrastructure will make an appropriate choice on behalf of the consumer. The infrastructure hides as many technicalities as possible from a requestor.

Particularly technical specificities from different implementation technologies such as J2EE or .NET should not affect the SOA users. (Endrei et al. 2004)

2.2 Software as a Service

A copy of software can be made available to consumers as a shared service accessible over network on demand. It changes the behavior of customers from buying a license to charging on subscription or pay-per-use basis. Business also finds out that subscription or pay-per-use mode of software charging more cost effective compared to purchase of software licenses and investing into infrastructure. (Namjoshi and Gupte 2009)

Turner et al. (2003) mention that SaaS separates the possession and ownership of software from its use and deliver functionality of software as a service which can overcome current limitations such as constructing software use and deployment for small-scale service providers and large organizations. The widely used service model is composed by two layers – supplier's software application service layer and service transport layer. Service transport layer uses forms such as .NET or J2EE to provide communication platform. Users use several protocols to construct services in Web services stack framework which is shown in Figure 1. They focus on three important protocols with different functionality – SOAP for communicating, web services description language (WSDL) for recording service descriptions and universal description, discovery and integration (UDDI) for discovering service.

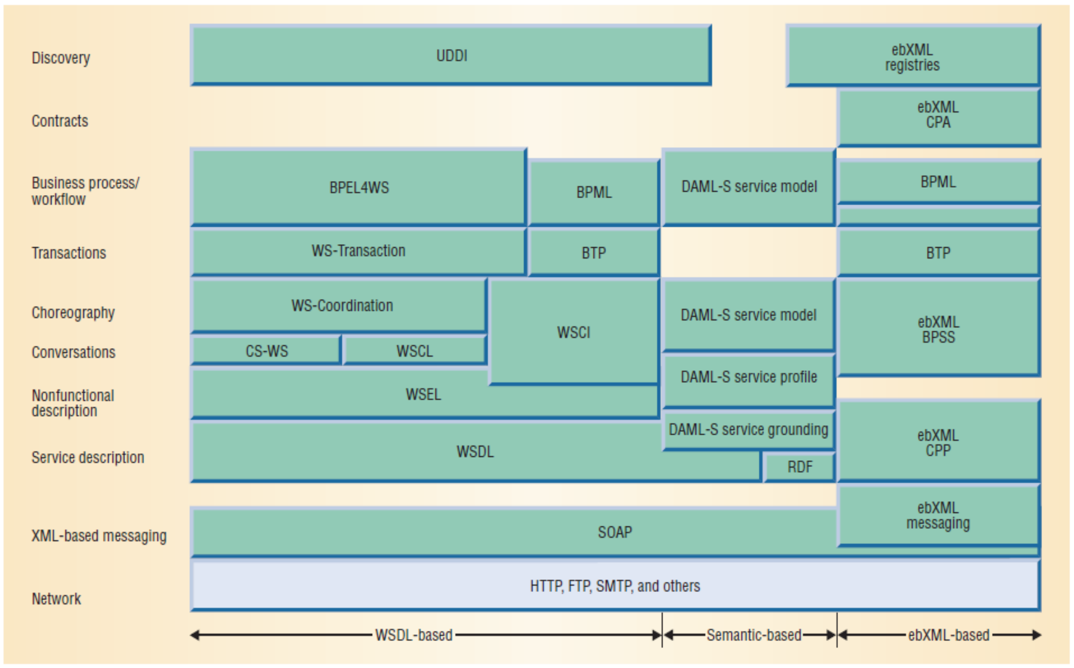

Figure 1: Web services stack framework (Turner et al. 2003)

As the concept of SOA and SaaS, the iPalace Video Channel can be part of a new service which combines other services in the future. But in the paper, we will only discuss the iPalace Video Channel.

2.3 Cloud computing

The cloud computing is one of the most popular terms in recent years and changes the environment of IT industry and the way of service constructing, innovating and delivering. Patterson et al. (2009) point out that developers with innovative ideas for new Internet services no longer require the large capital outlays in hardware to deploy their service or the human expense to operate it. They need not be concerned about overprovisioning for a service whose popularity does not meet their predictions, thus wasting costly resources, or underprovisioning for one that becomes wildly popular, thus missing potential customers and revenue. Moreover, companies with large batch-oriented tasks can get results as quickly as their programs can scale, since using 1000 servers for one hour costs no more than using one server for 1000 hours. This elasticity of resources, without paying a premium for large scale, is unprecedented in the history of IT.

The cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. There are lots of terminologies about the cloud computing which are shown in below:

- Software as a Service (SaaS): the applications delivered as services over the Internet
- Cloud: the datacenter hardware and software
- Public Cloud: a cloud is made available in a pay-as-you-go manner to the general public.
- Utility Computing: the service of the public cloud being sold such as Amazon Web service, Google AppEngine (Google AppEngine 2012) and Microsoft Azure (Microsoft Azure 2012).
- Private Cloud: an internal datacenters of a business or other organizations which are not available to the general public.

The cloud computing is the sum of SaaS and Utility Computing, but does not include Private Clouds. From a hardware point of view, three aspects are new in the cloud computing.

1. The illusion of infinite computing resources available on demand, thereby eliminating the need for Cloud Computing users to plan far ahead for provisioning.

2. The elimination of an up-front commitment by Cloud users, thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs.

3. The ability to pay for use of computing resources on a short-term basis as needed and release them as needed, thereby rewarding conservation by letting machines and storage go when they are no longer useful.

Utility computing has several types and is distinguished based on the level of abstraction presented to the programmer and the level of management of the resources. In general, there are three types which will be introduced below, including: Amazon Web service, Google AppEngine and Microsoft Azure. Amazon EC2 is at one end of the spectrum. An EC2 instance looks much like physical hardware, and users can control nearly the entire software stack, from the kernel upwards. There is no a priori limit on the kinds of applications that can be hosted; the low level of virtualization—raw CPU cycles, block-device storage, IP-level connectivity—allow developers to code whatever they want. On the other hand, this makes it inherently difficult for Amazon to offer automatic scalability and failover, because the semantics associated with replication and other state management issues are highly application-dependent.

At the other extreme of the spectrum are application domain-specific platforms such as Google AppEngine. AppEngine is targeted exclusively at traditional web applications, enforcing an application structure of clean separation between a stateless computation tier and a stateful storage tier. Furthermore, AppEngine applications are expected to be request-reply based, and as such they are severely rationed in how much CPU time they can use in servicing a particular request. AppEngine's impressive automatic scaling and high-availability mechanisms, and the proprietary MegaStore (based on BigTable) data storage available to AppEngine applications, all rely on these constraints. Thus, AppEngine is not suitable for general-purpose computing.

Microsoft's Azure is an intermediate point on this spectrum of flexibility vs. programmer convenience. Azure applications are written using the .NET libraries, and compiled to the Common Language Runtime, a language-independent managed environment. The system supports general-purpose computing, rather than a single category of application. Users get a choice of language, but cannot control the underlying operating system or runtime. The libraries provide a degree of automatic network configuration and failover/scalability, but require the developer to declaratively specify some application properties in order to do so. Thus, Azure is intermediate between complete application frameworks like AppEngine on the one hand, and hardware virtual machines (VMs) like EC2 on the other.

Patterson et al. (2009) argue that the construction and operation of extremely large-scale, commodity-computer datacenters at low cost locations was the key necessary enabler of Cloud Computing, for they uncovered the factors decrease in cost of electricity, network bandwidth, operations, software, and hardware available at these very large economies of scale. These factors, combined with statistical multiplexing to increase utilization compared a private cloud, meant that cloud computing could offer services below the costs of a medium-sized datacenter and yet still make a good profit.

Elasticity is another appealing factor for adopting cloud computing. Even though Amazon's pay-as-you-go pricing (for example) could be more expensive than buying and depreciating a comparable server over the same period, they argue that the cost is outweighed by the extremely important the cloud computing economic benefits of elasticity and transference of risk, especially the risks of overprovisioning (underutilization) and underprovisioning (saturation). The risk of mis-estimating workload is shifted from the service operator to the cloud vendor. These characteristics make the cloud an attractive infrastructure solution especially for web applications due to their variable loads. However, Christodorescu et al. (2009) also mention that the cloud has several drawbacks. First, application owners have little or no control of the underlying cloud infrastructure. Second, commodity machines are likely to fail more frequently. Any architecture design based on the cloud must handle machine failures quickly in order not to frequently disrupt service.

2.4 Load balancing

The load balance is a method to distribute workload across one or more servers, network interfaces, hard drives, or other computing resources. A load balancing appliance usually sits between the Internet and a server cluster. Clustering of web servers is a method of constructing scalable Internet services. As each new user request arrives, the load balancing appliance makes near-instantaneous intelligent decisions about the server best able to satisfy each incoming request. Load balancing optimizes request distribution based on factors like capacity, availability, response time, current load, historical performance, and administrative weights. (Phifer 2006; Adler 2010)

2.4.1 Hardware and software load balancing

The load balance can be applied by hardware and software. Wee and Liu (2010) say that the hardware-based load balance uses hardware-based components with specific applications to forward packets to different web servers for processing depending on the user session and the load on each web server. The hardware-based component is typically expensive. The hardware-based load balance is designed to handle high level of load, so it can easily scale. Because of cloud's commodity business model, hardware-based load balance is rarely offered by cloud providers as a service. Instead, one has to use a software-based load balance running on a generic server.

The software-based load balance distributes user requests by software and different software adopts a different algorithm. The software-based load balance uses additional software to monitor the load of nodes in a cluster. (Piórkowski et al. 2010; Wee and Liu 2010) For example, HAProxy (2012) is a software-based load balance. It's a free solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications. It implements an event-driven, single-process model which enables support for very high number of simultaneous connections at very high speeds.

HTTP protocol itself can also be used to perform load balancing. HTTP protocol has a built-in HTTP redirect capability, which can instruct the client to send the request to another location instead of returning the requested page. We can use HTTP redirect to redirect requests from front-end servers to a set of back-end servers to distribute requests and balance load of servers. (Fielding et al. 1999; Cherkasova 2000)

2.4.2 DNS load balancing

Another method of load balance is based on Domain Name System (DNS). The DNS load balancing maps a single hostname for the server to multiple IP addresses. The DNS load balancing returns a different IP address for the server with each DNS request, using, for example, a round-robin mechanism. (Moreno et al. 2011) The DNS load balance has two drawbacks. (Wee and Liu 2010) First, after a local DNS server caches the IP address information, all browsers contacting the same DNS server would get the same IP address and hence the load balance mechanism couldn't effectively distribute requests that come from a large number of hosts. Second, the local DNS server caches IP address for a period of time such as a few days. Unless the cache in the local DNS server expires, requests will be guided from browsers to the same web server. When variation of number of requests changes frequently and the period of time is not over days, like hours, the DNS load balancing has little effect.

Wee and Liu (2010) further concluded that the DNS load balance is not suitable for cloud-based servers. A cloud-based web server farm elastically changes the number of web servers tracking the volume of traffic in minute's granularity. Days of DNS caching dramatically reduces this elasticity. Another reason is that IP addresses for new web servers will not be propagated to DNS servers that already have a cached IP address. Therefore, even though the old web servers overload, those DNS servers will keep sending requests to them and the new web servers will remain idle.

2.4.3 Geographic load balancing

Many enterprises establish a worldwide Internet presence by deploying servers in many locations for the purpose of providing service to people around the world. To add scalability and disaster resistance, enterprises employ the geographic load balance. The geographic load balance (Phifer 2006) increases availability by allowing regional server clusters to share workload transparently and enables disaster recovery on a global scale. Even when everything is running smoothly, the mechanism to balance the load across regional server clusters offers many benefits. Response time can be minimized by directing clients to the closest server cluster. Google applies Content Distribution Network (CDN) to achieve geographic load balance. (Krishnan et al. 2009) CDN is a technique to improve client performance in client-server applications. Instead of hosting all contents on a server at a single location, content providers replicate their content across a collection of geographically distributed servers or nodes. Different clients are redirected to different servers both for load balancing and to reduce client-perceived response time. To reduce the response time perceived by clients in fetching Google's contents, the CDN redirects each client to the node to which it has the least latency. Latency-based redirection results in most clients being served by a geographically proximate node. But redirecting every client to the server with least latency does not suffice to optimize client latencies. Because of geographically nearby clients often see widely different latencies even though they are served by the same CDN node. And connections to most clients are impacted by significant queuing delays.

Another load balancing between multi-location is a location-agnostic, proportional load balancing strategy employed by Youtube (Youtube 2012). YouTube employs this proportional load balancing strategy among its seven data centers, where the proportionality is determined by the "size" of a data center, where the number of

IP addresses seen at each location is used as a rough estimate of the size of a data center. (Adhikari et al. 2010) This proportionality stays fairly constant over time and across different geographical locations, i.e., Points-of-Presence of the tier-1 ISP, and holds for both the client-to-YouTube and YouTube-to-client traffic.

2.4.4 Load balancing in the Cloud

Applications on the cloud tend to run on a cluster with many standard commodity web servers, thus requiring a scalable and agile load balancing solution. Load balancing in the cloud differs from classical thinking on load-balancing architecture and implementation. Adler (2010) evaluates the performances of several cloud-based load balancing solutions, including HAProxy, Amazon Web Services Elastic Load Balancer (Amazon ELB 2012), aiCache Web Accelerator (Aicache Web Accelerator 2012) and Zeus Load Balancer. Amazon Web Services Elastic Load Balancer provides a virtual load balancer with a DNS name. Requests which are sent to this hostname are delegated to backend web servers - the Amazon EC2 instances within the same geographical zone or within multiple zones in a geographic region defined by Amazon. Amazon Web Services Elastic Load Balancer also provides the capability to add or remove Amazon EC2 instances and check their stability**.** If it finds out unstable instances, it will automatically reroute traffic to stable instances until the unstable instances have been restored. (Liu and Wee 2009; Namjoshi and Gupte 2009)

The test retrieves a very small web page from backend servers via the load balancer under test. All tests are performed in the AWS EC2 US-East cloud, and all instances (application servers, server under test, and load-generation servers) are launched in a single availability zone. ApacheBench is used to generate load in the test. Five identically configured web servers were used in each test to handle the responses to the http requests initiated by the load-generation server. An architectural diagram of the test setup is shown in Figure 2. The load balancing test focuses on determining the maximum connection rate that each solution is capable of supporting. Adler (2010) states that “[p]articular use-cases may see more relevance in testing for bandwidth or other metrics, but we have seen more difficulties surrounding scaling to high connection rates than any other performance criterion.” (page 3) In the experiment, they use requests per second as the indicator of connection rate. Besides, they also monitor the interface traffic which combines incoming and outgoing packets on the load-generating servers and the number of Apache requests on the backend servers.

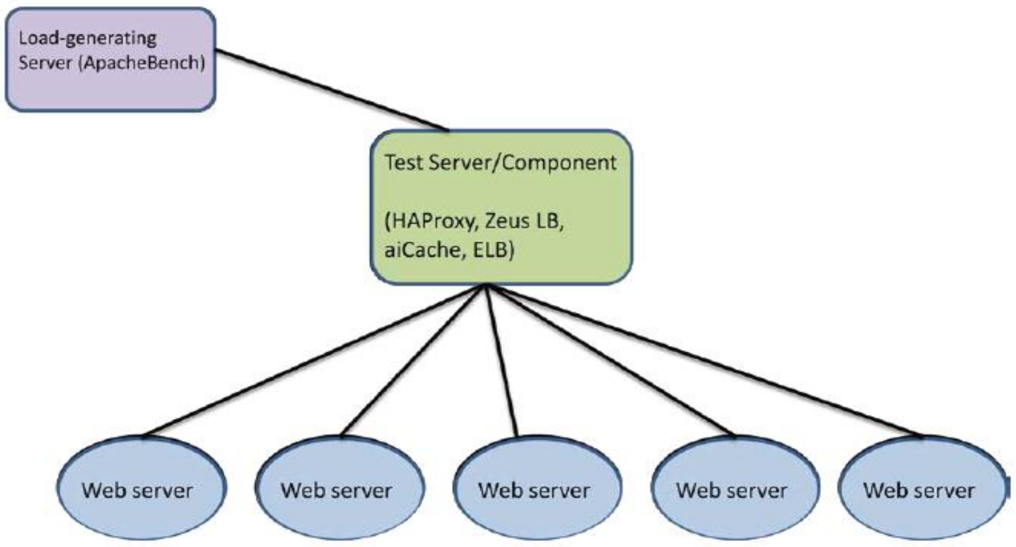

Figure 2: Modified Research Model (Adler 2010)

CPU utilization which includes all kind of CPU activity on all web servers was tracked during the tests to ensure this tier of the architecture was not a limiting factor on performance. The CPU idle value for each of the five web servers was consistently between 65%-75% during the entire timeline of all tests.

Adler (2010) concludes that “there is no ‘one size fits all’ when it comes to load balancing. Depending on the particular application’s architecture, technology stack, traffic patterns, and numerous other variables, there may be one or more viable solutions, and the decision on which mechanism to put in place will often come down to a tradeoff between performance, functionality, and cost.” The indicators are shown in Table 1. Requests per second has gathered in general tests for four cloud-based load balancing solutions and the rest of indicators are gathered in additional test of Amazon ELB and HAProxy.

Table 1: Indicators (Adler 2010)

Name	Description
Requests per second	the number of requests per second that were processed by the server under test
httperf responses per second	the number of httperf responses per second that were processed by the server in additional tests which were performed on the AWS ELB and on HAProxy using httperf as an alternative to ApacheBench.

CPU activity	CPU activity on typical backend servers, including: user, nice, system, interrupt, softing, wait, idle and steal. It can get CPU idle rate, too.
Interface traffic on the load-generating servers	Numbers of packets sent and received per second
Packets per second as generated by ttcp	Numbers of packets sent and received per second in additional test

Elasticity of cloud computing provides users more available on delivering services with stable quality, but users have to budget meticulously under pay-as-you-go business model. Ishii and Suzumura (2011) try to minimize the economic cost of using the cloud for data stream processing so that people under the limited budget also can apply load balancing in the cloud. Data stream processing is a way to support real-time processing of continuous data streams, including financial data analysis and data analyzing from large sensor networks. These applications need low latencies for real-time responses.

How to insure real-time processing is one of the key technical challenges in this area. One of the simple solutions is to add new physical nodes to guarantee that the overall system performance is adequate to handle the largest possible burst of data. However, various problems often prevent the use of this solution, such as budget limitations, inadequate electrical supply, or even space for the new hardware. Even if we can solve the problems and provide sufficient computational resources to handle the highest possible data rate, most of those resources may be wasted if the normal data rate is low. So, they construct the *ElasticStream* system on the cloud to solve this problem.

The ElasticStream system is an elastic data stream processing system that uses a cloud environment as needed by changing the number of computational nodes in the cloud. The application flow in the system can be broken down into three parts, first receiving the incoming data stream, then splitting the data up for multiple computational nodes, and finally processing it in parallel. The system also adds computational nodes in the cloud environment by spawning an appropriate number of VMs if the local environment is overloaded.

Whether or not they run the VM is a more important factor for calculating prices of computation. To calculate the required number of VMs, they predict the future average data rate and solve the optimization problem based on this prediction. The future data rate is predicted by the algorithm called SDAR (Sequentially Discounting AutoRegression). The steps are (1) to predict the future data rate, (2) to calculate the number of processes that can be handled by the local environment, and (3) to assign the remaining processes to the cloud environment. When the future data rate is larger than the amount of data that then local nodes can handle, then the system assigns the rest of the data stream to the cloud VMs, whose number is minimized to minimize the costs.

We build up the iPalace Video Channel on the cloud because the cloud platform has several benefits which discussed in the previous discussion of load balancing. The hardware-based load balance and the DNS load balancing is not suitable for cloud environment, thus, we use software-based load balancing on the cloud. Multi-location load balancing may be discussed in the future, and we do not consider the limitation of budget, so we will not discuss how to minimize cost.

We also set up an additional application to dynamically monitor and allocate VMs to handle the huge amount of requests of the iPalace Video Channel sometimes happened in a certain tiny period of time. We cluster VMs to handle bottleneck operations, and when the workload of the server with minimal workload in cluster is to a certain extent, we will invoke VM to share the workload. And we will close servers when the rest servers are capable to handle the requests.

2.4 Queuing model

Queuing model is the models and techniques to analyze situations whether or not to invest for reducing waiting time and the effect of the investment on the waiting time. (Adan and Resing, 2001)

There is a formal queuing system is shown in in Figure 3. There are several service counters or equipment for serving customers. When all the service counters is full-served, the customer who is waiting for being served will line up a waiting queue and the queuing system is formed. The customer will leave the queuing system when the service ends up.

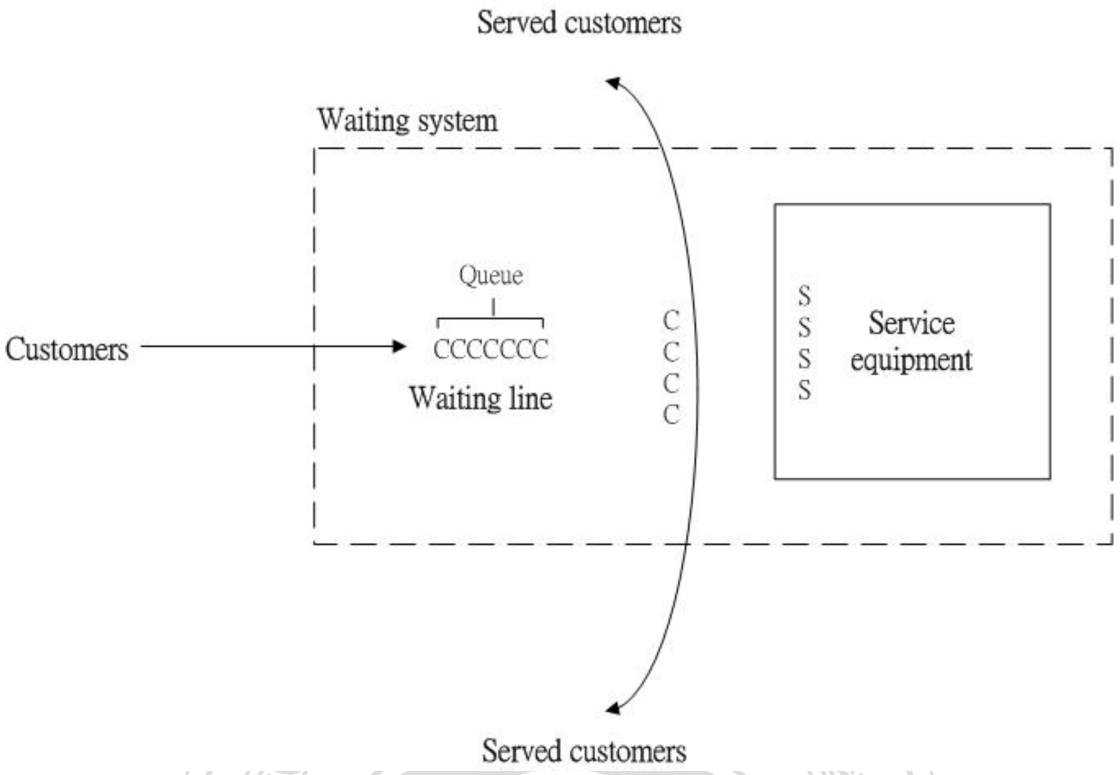

Figure 3: Queuing system

According to different situation, there will be a different queuing model. The queuing model is shown in Figure 4, the definition of the queuing system A/B/C : E/F/G is as follows:

- **A: The probability distribution of arrivals**

  M means the probability distribution of arrivals is Poisson distribution, E_k means the probability distribution of arrivals is Erlang distribution, D means the probability distribution of arrivals is deterministic distribution and GI means the probability distribution of arrivals is General distribution.

- **B: The probability distribution of the service time**

  M means the service time is Exponential distribution, E_k means the service time is Erlang distribution, D means the service time is deterministic distribution, and G means the service time is General distribution.

- **C: Number of Servers**

  1 means single line and N means multiple lines that the number of servers above 2.

- **D: Number of source in service system** number of source in service system includes length of waiting line and number of servers. N means finite and ∞ means infinite.

- **E: Input Source**

  N means the input source is finite and ∞ means the input source is infinite.

- **F: Queue Discipline**

  FCFS, LCFS, RSS and PR.

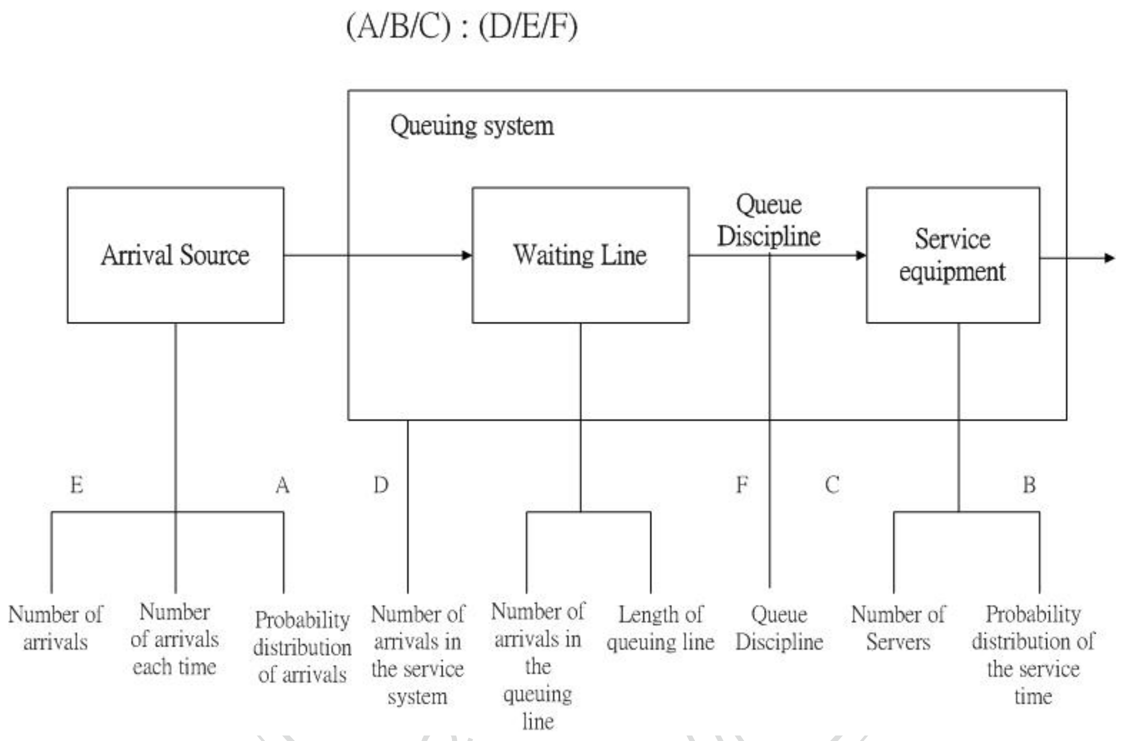
(A/B/C) : (D/E/F)
Queuing system
Arrival Source
Waiting Line
Queue Discipline
Service equipment
E
A
D
F
C
B
Number of arrivals
Number of arrivals each time
Probability distribution of arrivals
Number of arrivals in the service system
Number of arrivals in the queuing line
Length of queuing line
Queue Discipline
Number of Servers
Probability distribution of the service time

Figure 4: The queuing system A/B/C : E/F/G.

Except the load balance mechanism, we set up a two phase service system because the result of evaluation in section 5 shows that the load balance mechanism can distribute load to different VMs but the peak load in a short period still exist. The service system regarding the phase I is a M/D/1: ∞/∞/FCFS queuing system which accomplishes load balancing mechanism and satisfies the service quality. The service system regarding the phase II is a M/M/1: ∞/BOUND/round-robin queuing system which defines the constraint and helps us to figure out the best solution for satisfying service quality.

3. The iPalace Video Channel Service Design and Implementation

3.1 The iPalace Video Channel Service Design

Bilderbeek *et al.* (1998) state that the obtained service concepts, ideas, and criteria will affect the design of the new client interface and new service delivery system as well as technological options. The new service developing cycle of Johnson *et al.* (2000) also suggests that the corresponding service concepts, ideas, and criteria should be obtained and approved before proceeding to the design phase. Following these statements, the study first briefs the service concepts, ideas, and criteria regarding the iPalace Video Channel service.

Since video-based presentation can deliver more complete, more vivid, more tele-presenting, and more appealing information to viewers than text-based presentation (Hoffman and Novak, 1996), the NPM dreams of a video-based website that makes people around the world be capable of accessing the video collection of Chinese relics in NPM. Furthermore, the NPM wants to outreach the young generation via providing them wonderful viewing experiences, such as aesthetic pleasure, culture, intellectual, creative and so forth to enhance their quality of life. And in the near future, the video-based service is better to be accessed via portable devices (like tablet PCs and smart phones). In addition, in the age of information explosion, the key to winning for web service is not about providing more information, but about how to systematically finish and distinguish valuable information for browsers (Nobel, 2011). Therefore, the iPalace video channel service should systematically collate and present the treasures of NPM, enabling users to experience NPM's rich art collection methodically. These goals lead to an initiative of the iPalace Video Channel service that is customer-oriented and innovative, and whose quality should meet the NPM's brand image.

Table 2 shows the strategic vision of the iPalace Video Channel service. The target customer segment of the iPalace Video Channel service is the young generation that can access World Wide Web, loves Chinese heritage, and likes watching video more than reading text. One of service concepts is to provide the customer an impressing viewing experience of NPM artifacts. One of key ideas regarding this service concept is to follow the concept of museum exhibition instead of the concept of search-in-warehouse to well organize the videos. Another key idea is to make the video service available anytime and anywhere. In addition, the video service should be provided via an uninterrupted multicasting way. Another service concept is to make sure that the user interface (UI) is ease-of-use and has profound senses of Chinese culture and aesthetic pleasure.

Table 2: The strategic vision of the iPalace Video Channel service.

Service Delivery System	Operating Strategy	Service Concept	Target Market Segments
✧Fresh and attracting video contents ✧Well-defined experience that ● meets with the NPM image ● is ease-of-use ● has profound senses of Chinese culture and aesthetic pleasure ✧Smooth video delivery	✧Continually providing new video exhibitions ✧Periodically changing the interface appearance ✧Effectively load balancing through ● Task-oriented process design ● Elastic web service infrastructure ● peer-to-peer networking ● multi-task and distributed system architecture	✧The video service of smoothly displaying NPM relics that is ● Well-arranged exhibition ● Available anytime and anywhere ● Uninterrupted multicasting ✧The user interface is ease-of-use and has profound senses of Chinese culture and aesthetic pleasure	✧The young generation that ● is web browser ● is interested in Chinese heritages ● enjoys to watch video

The operating strategy is to keep the website fresh and attracting for the customer retention and to effectively load-balance for smoothly displaying videos even under the huge peak-demand from worldwide. To keep the website fresh and attracting requires continually providing new video exhibitions and periodically changing the interface appearance. For effectively load-balancing, the iPalace Video Channel service technically requires the task-oriented process design, the elastic web service infrastructure, the peer-to-peer networking, and the multi-task and distributed system architecture. The service delivery system should always smoothly deliver fresh and attracting video contents and well-defined NPM experiences.

The customer criteria for embracing the iPalace Video Channel service include the reputation of NPM brand, the dependability and education of contents, the high-quality video, the ease-of-use and aesthetic UI, the stable and smooth Web service accessible worldwide and free-of-charge. In short, being an online media of NPM, the iPalace Video Channel service needs to not only meet with the high-quality brand image of NPM but also be able to cope with the huge peak-demand from worldwide.

3.2 Implementation of the iPalace Video Channel

Technically, the iPalace Video Channel is a marketing platform in Internet that provides video-stream multicasting, video-on-demand, and information retrieval services, and so on. We build up the iPalace Video Channel on the cloud to provide the video service.

Ciuffoletti (2010) describes a basic elastic web service infrastructure on the cloud as shown in Figure 5. Physical machines in the infrastructure management layer such as server arrays, storage and routers are provided by cloud computing service providers. The iPalace Video Channel is built as the user application layer on top of the infrastructure management layer that is provided by Chunghwa Telecom's Hicloud. (Hicloud 2011) The service is implemented in the iPalace Video Channel by using high-level language technologies and APIs. General users won't connect to the infrastructure management layer directly but can access services from the iPalace Video Channel.

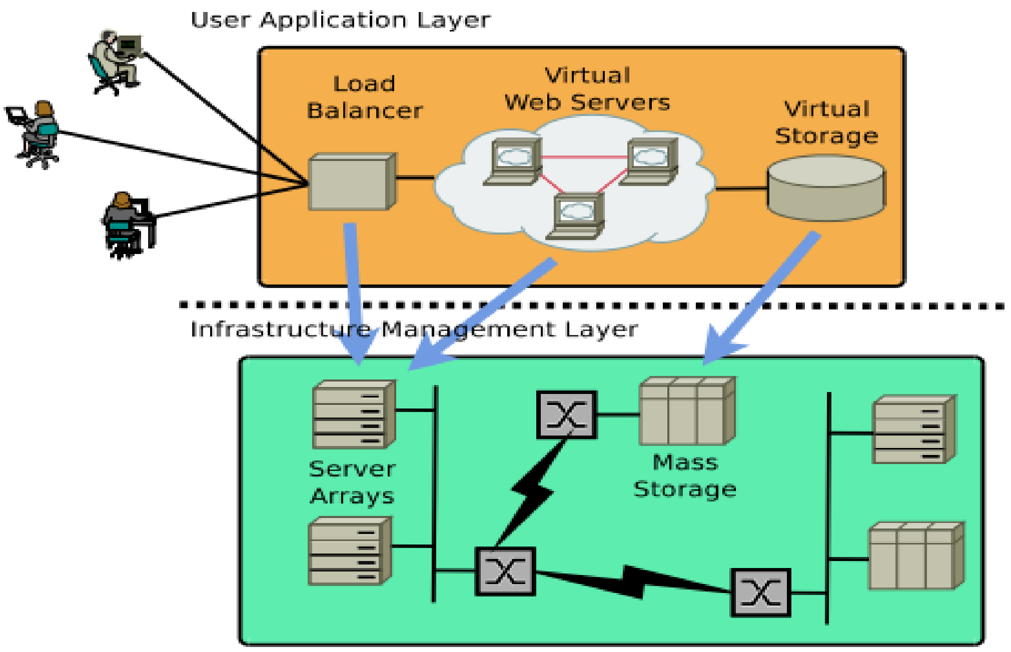

Figure 5: infrastructure and user application views of a basic elastic web service infrastructure (Ciuffoletti 2010)

As shown in Figure 6, the structure of the iPalace Video Channel that has a multi-task and distributed system architecture. We build up servers on VMs separately according to the tasks they are responsible for. Tasks are split and distributed to each VM so that we can not only implement each VM efficiently but also avoid centralizing workload on a particular VM. Since each VM takes a specific task, this makes distributing requests among a cluster of identical VMs much easily.

The structure of the iPalace Video Channel consists of 4-tier layers. Each VM has 2 virtual cores (2GHZ), 4G memories, and runs the CentOS 5.5, 64bit version. All the VMs have installed the Apache server package. Layer-1 contains the Main Reception Desk that is responsible for receiving requests from users who would like to access the iPalace Video Channel service. The main reception disk hosts the index webpage and bypasses the requests to the next layer for accessing videos. Layer-2 is composed by six distribution centers (DCs) that contain streaming servers and (local-cached) videos. All the DCs have installed the flash media server (the streaming server) and are responsible for broadcasting videos to users. DCs are classified into the following the video categories: main, AD (i.e., advertisement), (the introduction of) NPM, mission (i.e., the introduction of the iPalace Video Channel), collection (of historic relics) and How-to-use (i.e., the guideline of the iPalace Video Channel). The DC Main is responsible for distributing requests to different DCs by sending their index and needed information back to users. After receiving the index, all the following user requests will be redirected to the indexed server (that corresponds to the service). The index is selected according to the category of videos chosen by users. Later we extend each DC to an elastic cluster of VMs and cope with the high peak-demands by balancing the load.

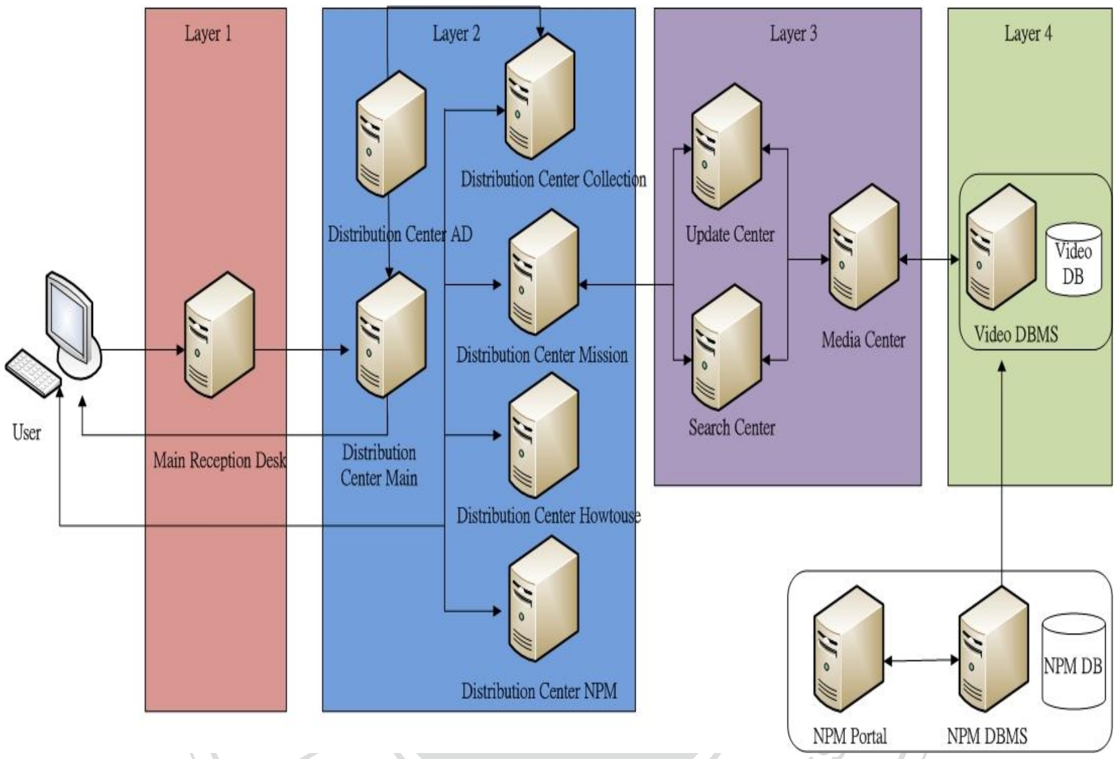

Figure 6: The system structure of the iPalace Video Channel

The backstage operations are supported in the next two layers. Layer-3 contains the Update Center, the Search Center and the Media Center. The Media Center is responsible for connecting Video DBMS when searching and updating videos. The Update Center checks with the Media Center to update new uploaded or deleted videos. The Search Center delivers the search request from the previous layer to the Video DBMS through the Media Center and returns the result to the users. The Video DBMS and Database belong to Layer-4, within which the NPM employees can upload new videos from the portal and back up videos that have been uploaded in a local NPM DB.

In Figure 7, we demonstrate the flow of the process for user requesting for the iPalace Video service.

(1) User sends requests for requesting the iPalace Video service to the Main Reception Desk.

(2) The Main Reception Desk wraps the user requests with related information and directs to the DC Main.

(3) The DC Main returns the index webpage to the user

(4) After receiving the response, all the following user requests will be redirected to the DC according to the category of videos user chosen and establish connection. Then, the DC provides the asked service, e.g., the NPM collection series as shown in Figure 7.

(5) On the other hand, during the idle time of the iPalace Video Channel service (e.g., users do not ask for other videos), the DC AD will broadcast non-stop advertisement automatically to the users.

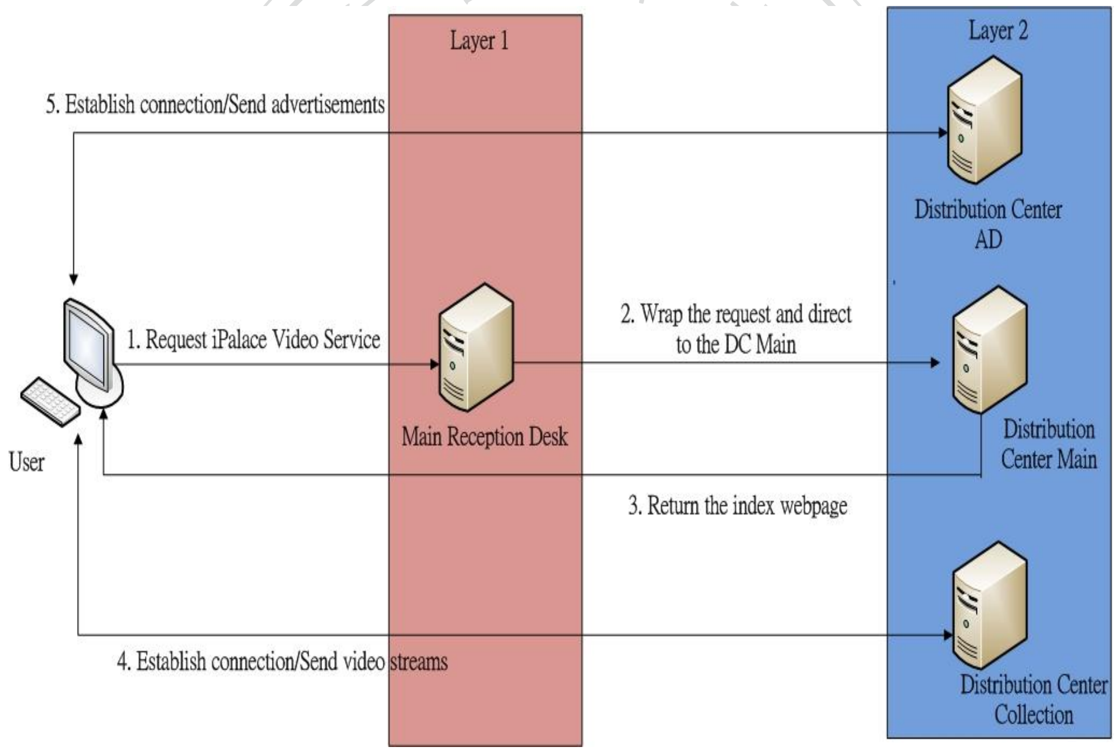
Layer 1
Layer 2
5. Establish connection/Send advertisements
Distribution Center AD
1. Request iPalace Video Service
2. Wrap the request and direct to the DC Main
Main Reception Desk
Distribution Center Main
User
3. Return the index webpage
4. Establish connection/Send video streams
Distribution Center Collection

Figure 7: Process of coming on in the iPalace Video Channel for the first time

We develop the task-oriented VMs separately according to the service and video category. Despite offering a clean task to each VM, the design may not be able to disperse the workload on servers in balance. Particularly, the VMs for DC AD and DC Collection may take most workload since the services appear to be requested frequently by users and systems. In fact, one aim of the iPalace Video Channel service is providing a non-stop multi-casting video service stored in the DC AD and the DC

Collection. The potentially high peak overload of the VMs on the DC AD and the DC Collection could lead to an unsatisfactory experience of the user. In the next section, we will describe how to build an elastic cluster of VMs to tackle the potentially high peak overload of the VMs on the DC AD and the DC Collection with the aim of providing a consistent high-quality service.

In the following, we describe how to dynamically allocate VMs to achieve the load balance with satisfied service quality. The main idea is to employ an elastic cluster of VMs to take and share the workload. A cluster consists of a set of identical VMs[3]. Unlike all the requests are taken by a single machine, requests are distributed in balance to each machine in the cluster. In addition, the size of the cluster is dynamically increasing so that the maximal load of all VMs within the cluster is low enough to guarantee an acceptable service quality.

Initially, the cluster has only one VM which is prepared to handle requests. We set up an auxiliary monitor VM – the Monitor watching the workload of all the VMs within the cluster. The Monitor periodically detects the number of arrivals[4] served by each running VM and reports the VM that has the minimal number of arrivals. If its number of arrivals is under an acceptable bound, it will be selected as the corresponding VM and the next coming arrival(s) will be dispatched to this VM. On the other hand, if it exceeds the bound (so as other VMs), we will increase the size of the cluster by initiating a new VM[5] to join the cluster and dispatch the next arrival to this VM. The new invocation is needed since, when the minimal one equals the threshold, it implies that all the other VMs also have equaled number of arrivals as the threshold. The new VM has the same ability to handle user's demands as the original VM. By the mechanism, we can achieve load balancing among an elastic cluster.

Except the Monitor, we set up another auxiliary VM – the Dispatcher which is responsible for dispatching the arrivals to the corresponding VM. The Dispatcher will retrieve the corresponding VM identified by the Monitor periodically and store it in an internal variable. We define *DURATION* as the interval between the Dispatcher retrieves the corresponding VM. DURATION determines the frequency of updating which VM takes the coming arrivals. The smaller the DURATION, the more often the retrieve of the corresponding VM is, so more balanced the load is.

[3] The identical VM has the same ability to handle the demanding video service as the others.

[4] Here an arrival corresponds to an initial service request from one individual User.

[5] The new VM has the same ability to handle the arrival's demands as the other running VMs. This can be achieved by initiating a new VM with the same image or simply by invoking an idle VM that was built with the same image.

Since we distribute requests by looking up the corresponding VM, all the requests come within the period of DURATION will be assigned to the corresponding VM. That is, if requests increase suddenly within the DURATION, it is possible the corresponding VM gets overloaded. Thus, for the purpose of preventing violating the requirement of acceptable service quality, if the number of arrivals that the corresponding VM handles with counted by the Dispatcher equals the *BOUND* –the upper bound of the number of arrivals that each VM can handle with, the Dispatcher will initiate a new VM, add it to the cluster, and update the corresponding VM stated in the internal XML file of the Monitor to select it as the corresponding VM.

When the cluster becomes larger, there may be more unemployed VMs which have no live arrivals in the cluster. To prevent wasting recourses, the unemployed VMs will be released to an *idle VM pool* which collects VMs that have been initiated but are not handling any arrivals. While the Monitor or the Dispatcher needs to initiate a new VM, it is randomly selected from this idle VM pool. Since all VMs are identical (in functions and status), getting a new VM from the idle VM pool can prevent the time of initiating a new VM from scratch. Furthermore, we keep the number of VMs in the idle VM pool in an acceptable range. When the number of VMs is out of the range, extra VMs will be released (or initiated) automatically.

The Monitor also gathers other status information of each running VM periodically including: The average of CPU load, the number of received and sent packets per second, the number of requests, the average connections etc. This information provides us the ability to monitor abnormal behaviors and adjust the cluster accordingly.

In the following we detail how we monitor the VMs, select the VM with the minimal number of arrivals and update the internal XML file (for machines to access the current minimal load VM) in the Monitor. The pseudo code of the algorithms is given as below:

Algorithm *monitor_handler()*

```
1.  define BOUND;
2.  init hostname[], log;
3.  while true:
4.      set_xml(monitor(hostname[], log, BOUND));
```

Algorithm *monitor(*hostname[], log, BOUND*)*

```
1.  init load[]= init_load (hostname[]);
2.  monitor_VMs (hostname[], load[], log);
```

```
3.  minhost = select_min(hostname[], load[]);
4.  if(load[minhost] > BOUND)
5.          new = create_VM();
6.          minhost = new;
7.          update(hostname[], new);
8.  return minhost;
```

The control variable BOUND specifies the upper bound of the number of arrivals that each VM can handle with. It also determines when to new a VM to increase the size of the cluster, which has an influence on the service quality.

The array hostname is used to record the names of VMs in the cluster. It is set with one VM by default. While a new VM is created (monitor(), line5-7), the hostname is updated with the new VM. When calling the procedure monitor_VMs(), we connect all the VMs listed in the hostname and collect their status, such as the average of CPU load over 1 minute from /proc/loadavg and the number of connection established (saved in load[]). We also collect some other data related to the CPU and network status of each VM in the log using three unix commands – uptime, iostat and netstat, as well as requests per second (collected from the Apache's server status). Uptime provides: the current (logged) time, how long the system has been running, how many users are currently logged on, and the system load average for the past 1, 5, and 15 minutes. Iostat is used for monitoring system input/output device loading by observing active devices in relation to their average transfer rates. It also reports the CPU and the device utilization. Netstat provides information about the linux networking subsystem, e.g., a list of open sockets. The information (log) is used for evaluating and adjusting our load balancing strategy.

The presented strategy is selecting the VM with the minimal number of arrivals as the corresponding VM to take the coming requests before the next monitoring process. If the value of the minimal one equals the threshold (BOUND), we will initiate a new VM (by selecting one from the idle VM pool) to share the workload. By calling the procedure set_xml(), we update the XML file with the IP of the corresponding VM. The coming requests will be directly redirected to the VM specified in the XML file without any new monitoring process. Except in the monitor procedure, the new VM will be initiated and selected as the corresponding VM and the XML file stated in the Monitor will be updated when number of arrivals that the corresponding VM handles with equals BOUND. In the end of the monitor procedure, the unemployed VMs will be released to the idle VM pool. In this way, we prevent run-time overload and resource wasted.

4. The Service System of the iPalace Video Channel

In this section, we present the service system in formal. Without loss of generality, we use one service, i.e., the DC Collection, as the clustering target. Other DC services can be dealt in a similar way. The architecture of a clustering target is shown in Figure 8. A cluster of VMs (instead of a single VM) is employed to serve the request of the video service of DC Collection. As shown in Figure 8, there is a monitoring VM that executes the monitoring procedure. Within each monitoring process, the Monitor finds the corresponding VM that handles with the minimal number of arrivals among all the current running VMs in the cluster. If the number of arrivals that the corresponding VM handles with is equal to the BOUND, an upper bound on the number of arrivals that each VM can handle with, the Monitor initiates a new VM by selecting one from the idle VM pool, adds it to the cluster, and selects it as the corresponding VM. In this case, the size of the cluster will increase one. The new invocation is needed since, when the minimal one equals the BOUND, it implies that all numbers of arrivals that the other VMs handle with also equals or exceeds the BOUND. At the end of each monitoring process, via calling the procedure set_xml(), the Monitor updates the internal XML file with the IP of the corresponding VM. The coming arrivals will be directly redirected by the Dispatcher to the VM specified in the XML file. Note that the BOUND determines the time epoch of initializing a new VM to increase the size of the cluster of running VMs.

There is no blank interval between each monitoring process, and the execution time of each monitoring process is different according to the number of VMs in the cluster. The more VMs in the cluster, the longer execution time of monitoring process is. As we mentioned earlier, DURATION is the interval between the Dispatcher retrieves the corresponding VM stated in the XML file in the Monitor that determines the frequency of updating the corresponding VM for the Dispatcher. The smaller the DURATION is, the more often the retrieving of the corresponding VM stated in the XML file. Even though within the DURATION, if the number of arrivals that the corresponding VM handles with equals the BOUND, the Dispatcher will initiate a new VM and select it as the corresponding VM. Later, we show that the BOUND is a control variable that may influence the service quality.

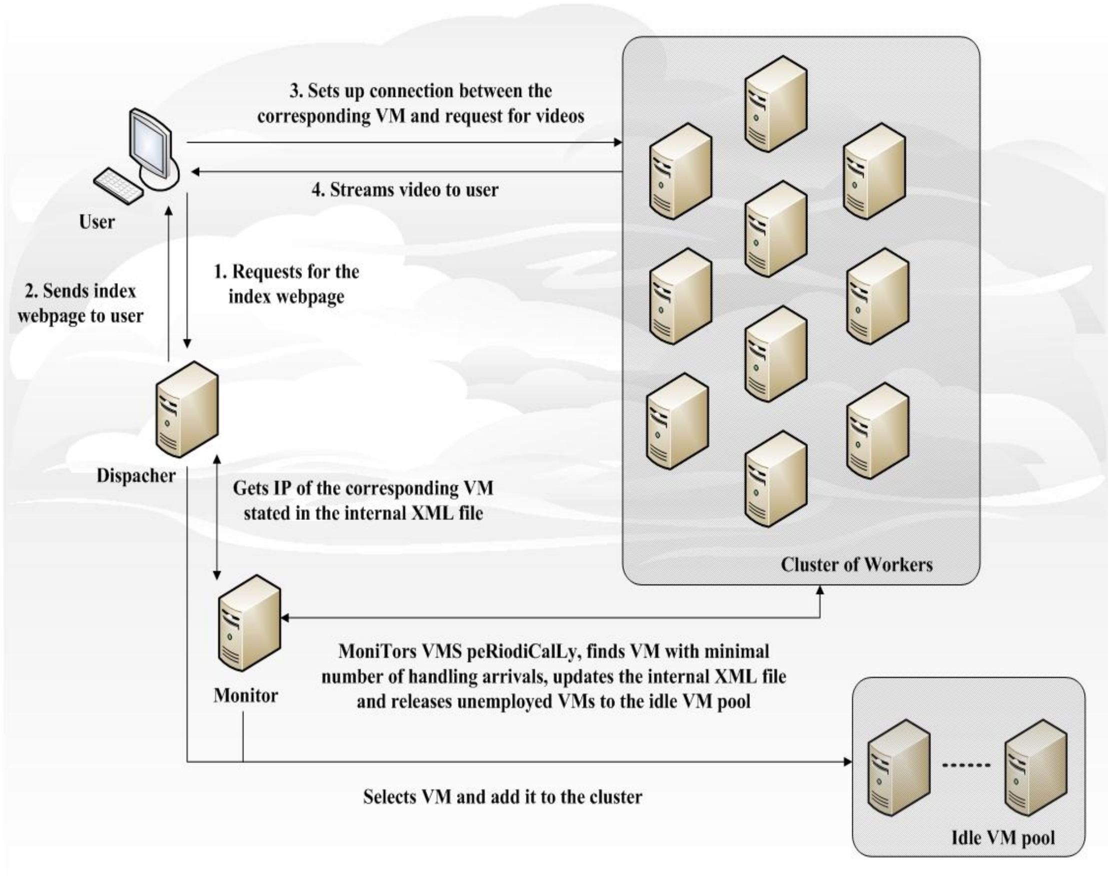
3. Sets up connection between the corresponding VM and request for videos
4. Streams video to user
User
2. Sends index webpage to user
1. Requests for the index webpage
Dispacher
Gets IP of the corresponding VM stated in the internal XML file
Cluster of Workers
Monitor
MoniTors VMS peRiodiCalLy, finds VM with minimal number of handling arrivals, updates the internal XML file and releases unemployed VMs to the idle VM pool
Selects VM and add it to the cluster
Idle VM pool

Figure 8: An elastic architecture for delivering the video service regarding the DC collection.

As shown in Figure 8, the process of delivering the video service regarding the DC Collection consists of four steps that can be grouped into the following two phases:

PHASE I:

(1) The User sends the initial request to the Dispatcher of the cluster of DC Collection.

(2) The Dispatcher responds with the index webpage to the User.

PHASE II:

(3) Via setting up the connection between the User and the corresponding VM, the index webpage explicitly directs the User to the corresponding VM.

(4) The corresponding VM provides the subsequent video service by infinitely broadcasting the selected video streaming of the DC Collection that interweaves with the advertisements (similar to a television program) until the User issues a new service request.

In short, in the phase I, the Dispatcher dispatches all initial user requests to their corresponding VMs that take care of the subsequent video service. In the phase II, each User receives non-stop video streams from the corresponding VM.

The service system regarding the phase I is a multichannel, single-phase service system with a variant amount of VMs such that every arrival is served by a VM and there is no balking. With the following assumptions, the service system regarding the phase I is a M/D/1: ∞/∞/FCFS queuing system[6]:

1. Arrivals are served on a FCFS (first come, first serve) basis.
2. Arrivals are independent of preceding arrivals, but the arrival rate λ does not change over time.
3. Arrivals are described by a Poisson probability distribution and come from a very large population.
4. Service times for all arrivals are fixed and known, says *d*.

That is, the arrival process is a Poisson process with mean rate λ; the service time is fixed as *d*; and the size of the requesting population is finite, but very large.

In this M/D/1: ∞/∞/FCFS queuing system, the service time is fixed as *d*. Thus, to have a stable system, λd needs to be smaller than 1. Furthermore, the steady-state probability of zero arrivals in the system P_0 equals $1 - \lambda d$; The average number of customers in the service system L equals $\frac{2\lambda d-\lambda^2 d^2}{2(1-\lambda d)}$; the average number of arrivals waiting in queues L*q* equals $\frac{\lambda^2 d^2}{2(1-\lambda d)}$; the average time spent in the service system W equals $\frac{2d-\lambda d^2}{2(1-\lambda d)}$; and the average time spent waiting in queues W*q* equals $\frac{\lambda d^2}{2(1-\lambda d)}$.

Without violating the requirement of acceptable service quality, the load balancing strategy is to select the running VM that has the minimal number of arrivals or to initiate a new VM as the corresponding VM to handle with the requests coming within each period of DURATION.

Note that, in step (4) of Figure 8, the designed service time for each arrival is infinite. However, due to the User's reneging behavior in choosing an incomplete service, the real service time X for each arrival is a random variable and finite.

[6] The definition of the queuing system A/B/C : D/E/F please refer to Appendix.

Furthermore, in the phase I, all arrival requests that come within the period of DURATION are dispatched to the same corresponding VM. But, in order to insure the high standard quality of service, each corresponding VM will be assigned to handle with at most the BOUND number of arrivals. Thus, the Dispatcher will count the number of arrivals that the corresponding VM is handling. If the number of handling arrivals equals to BOUND, the Dispatcher will initiate a new VM, add it to the cluster, and update the corresponding VM in the internal XML file to select it as the corresponding VM. The new arrivals will be dispatched to the corresponding VM that just be selected.

The service system regarding the phase II is a M/M/1: ∞/BOUND/round-robin queuing system. Assumptions of this service system regarding the phase II are as follows:

1. Arrivals are served on a round-robin basis.
2. Arrivals are independent of preceding arrivals, but the arrival rate λ_2 does not change over time.
3. Arrivals are described by a Poisson probability distribution and come from a very large population.
4. Service times also vary from one arrival to the next and are independent of one another, but their average rate μ (which is equal to $\frac{1}{E(X)}$) is known.
5. The service time X occurs according to the negative exponential probability distribution so that $P(X \leq x) = 1 - e^{-\mu x}$ for $0 \leq x$.
6. Each running VM serves at most the BOUND amount of arrivals.

That is, the CPU of the VM uses the round-robin algorithm to handle queuing tasks of all arrivals; the arrival process is a Poisson process with mean rate λ_2; the service time has an exponential distribution with mean rate μ; the size of the requesting population is finite but very large; and each running VM serves at most BOUND arrivals. Since the service system regarding the phase I is stable and has steady states, λ_2 should be equal to λ. The BOUND is the decision variable to make sure that the service quality for every arrival meets the predetermined level.

The round-robin algorithm is designed especially for time-sharing systems. The predefined time quantum is a small unit of time. In general, a time quantum is from 10 to 100 milliseconds. The CPU scheduler goes around the ready queue and allocates the CPU to each task for a time period of one time quantum. Arrivals enter and join a first-come-first-serve (FCFS) queue. The CPU scheduler picks the first task from the ready queue, sets a timer to interrupt after one time quantum and dispatches the task. (Silberschatz 2004) As shown in Figure 9, if the selected arrival's tasks are not finished at the end of a predetermined time quantum, the remainder of the tasks is effectively fed back to the end of the queue. (Rasch 1970)

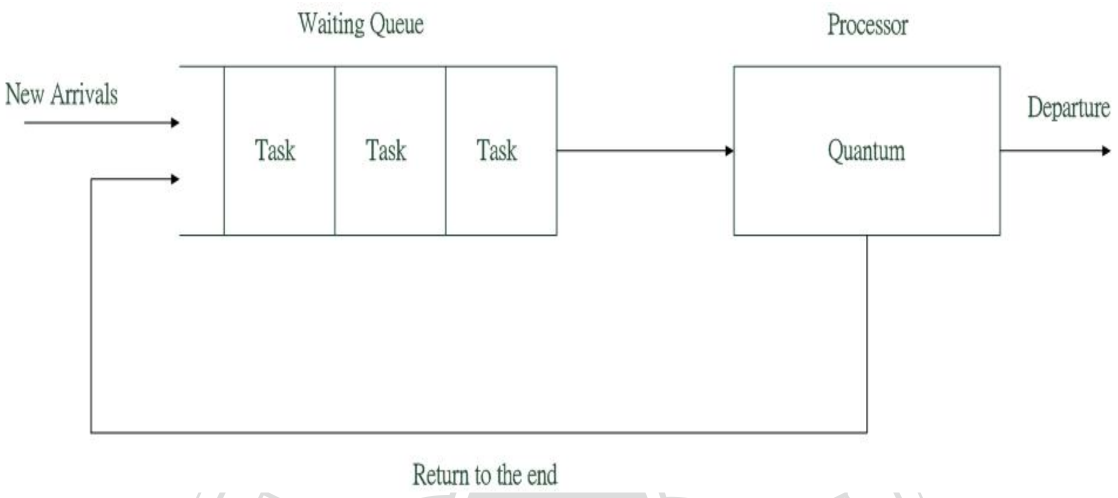

Figure 9: The queuing model with round-robin discipline

For example, there are three arrivals Arrival 1 to Arrival 3 which are shown in Figure 10. The task of each arrival is divided into several sequential pieces that are named in order. Because arrivals are served on a round-robin basis, tasks will be handled as the sequence T1.1, T2.1, T3.1, T1.2, T2.2, T3.2, T1.3, and so on.

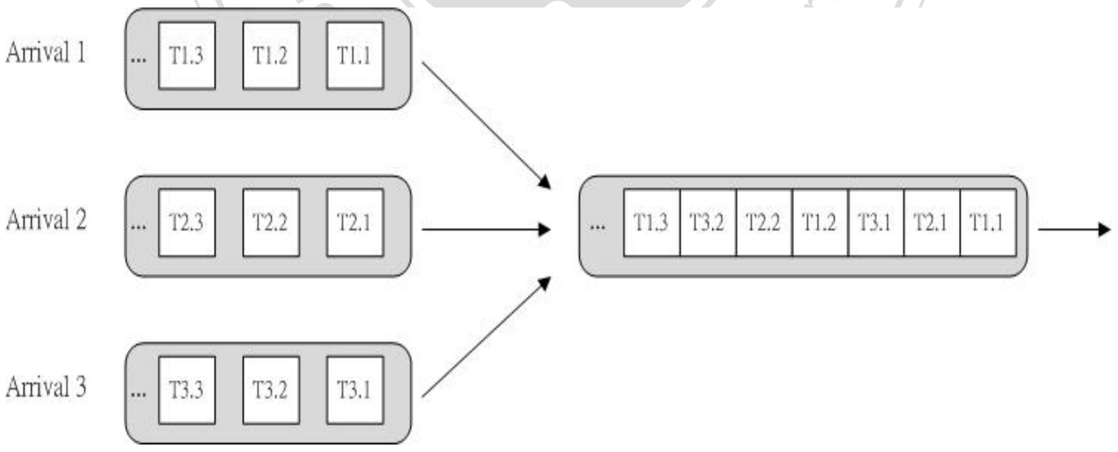

Figure 10: The schedule of three arrivals (Silberschatz 2004)

Figure 11 shows the schedule with new coming arrivals. At first, there are three arrivals in queue and the new Arrival 4 lines into the service system after the first round of handling tasks. Tasks in Figure 13 are handled by the following sequence: T1.1, T2.1, T3.1, T1.2, T2.2, T3.2, T4.1, T1.3, T2.3, T3.3, T4.2, and so on,

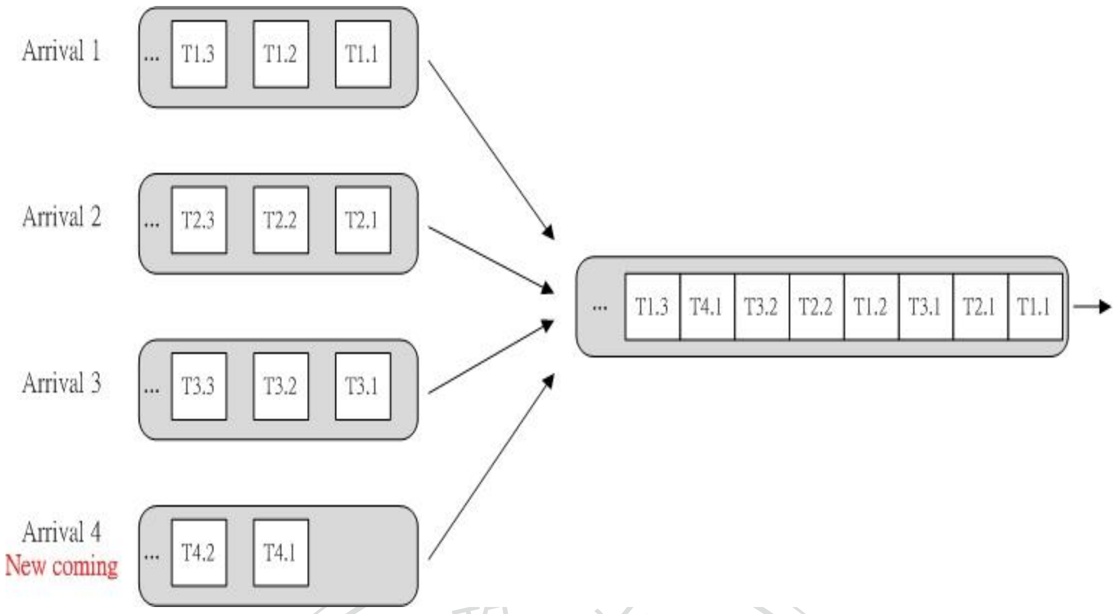

Figure 11: The schedule with new coming arrivals

Different from a regular queue, the User may renege from receiving the video service anytime and thus choose an incomplete service. Figure 12 shows the schedule scenario in which the Arrival 2 reneges after the first round of handling tasks. Tasks in Figure 12 are handled by the following sequence: T1.1, T2.1, T3.1, T1.2, T3.2, T1.3, T3.3, and so on,

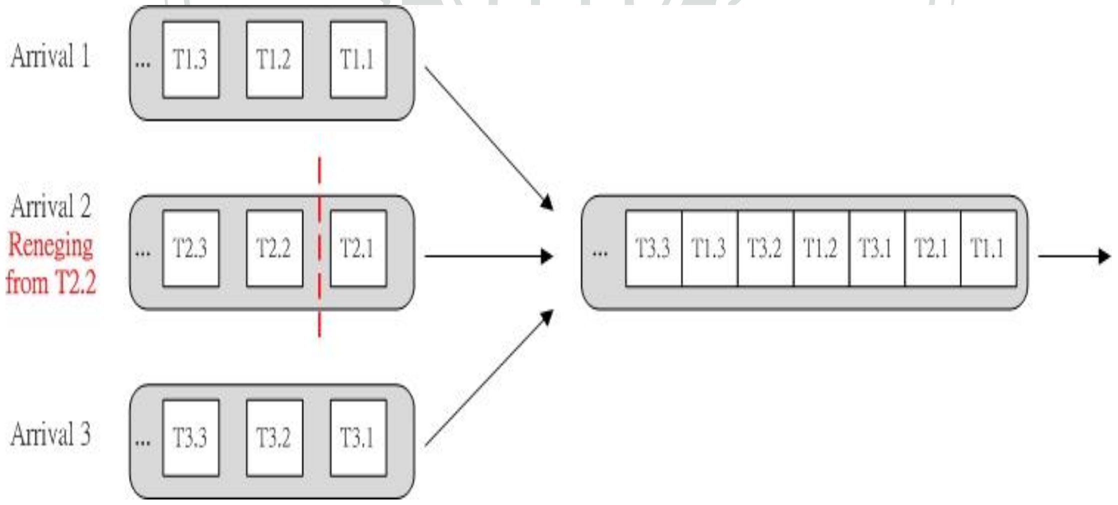

Figure 12: The schedule with reneging arrivals

As shown in Figure 13, the CPU spends a fixed time period (a time quantum) in handling the task of certain arrival, while the remaining tasks of the other arrivals are waiting in queue without being executed. That is, tasks of each arrival are taken iteratively and thus the service/non-service time appears rotationally for each arrival. The non-service time will change according to the number of arrivals in the queue.

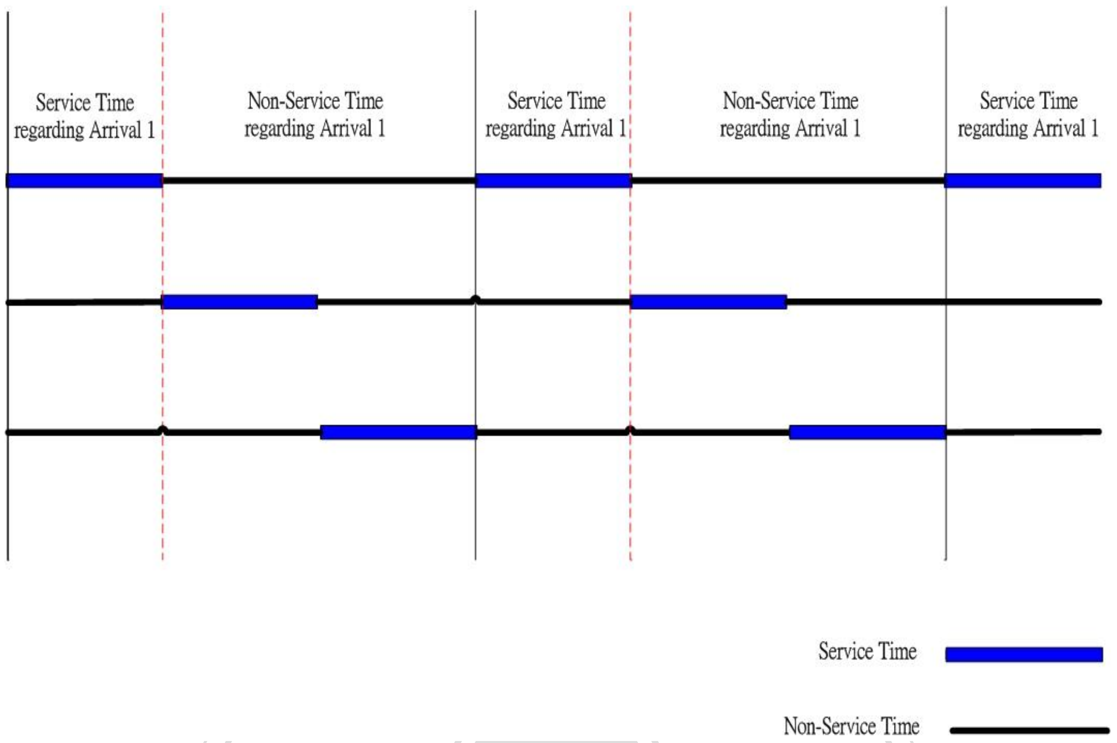
Service Time regarding Arrival 1
Non-Service Time regarding Arrival 1
Service Time regarding Arrival 1
Non-Service Time regarding Arrival 1
Service Time regarding Arrival 1
Service Time
Non-Service Time

Figure 13: schedule of handling three tasks

Figure 14 shows the state transition diagram of the queuing system regarding the phase II. In general, there are arrivals coming with an arrival rate λ and reneging with a reneging rate μ that equals $\frac{1}{E(X)}$. We define that the system is at the i th state when there are i existing arrivals observed in the system and P_i is the corresponding steady-state probability. Note that, in the i th state, the arrival comes on in the system in the arrival rate λ and the corresponding service rate equals the number of existing arrivals *i* multiplies the renege rate μ since each existing arrival has a service time X whose E(X) equals $\frac{1}{\mu}$.

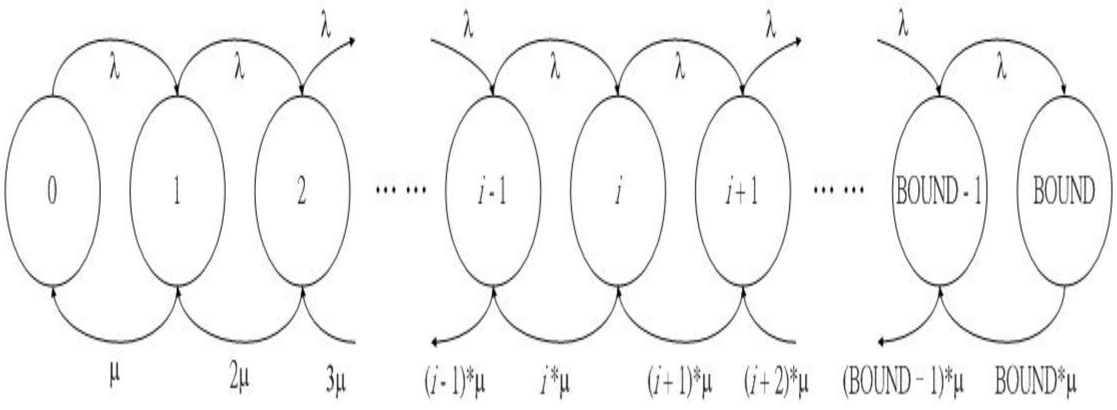

Figure 14: The state transition diagram of the queuing system regarding the phase II.

In a steady-state queuing system, the number of coming arrivals should be equal to the number of reneging arrivals for each state and we can calculate the steady-state probability via the balance equation (that is, in rate = out rate). (Lipsky 2009)

From Figure 14, we have the balance equations in the equation (1) regarding the queuing system of the phase II.

$$\begin{cases} P_0 * \lambda = P_1 * \mu \\ P_0 * \lambda + P_2 * 2\mu = P_1 * (\lambda + \mu) \\ \vdots \\ P_{i-1} * \lambda + P_{i+1} * (i+1)\mu = P_i * (\lambda + i\mu) \\ \vdots \\ P_{BOUND-1} * \lambda = P_{BOUND} * \mathrm{BOUND} * \mu \end{cases} \tag{1}$$

We can solve the equation (1) to get

$$\begin{cases} P_1 = \frac{\lambda}{\mu} * P_0 \\ P_2 = \frac{\lambda^2}{2\mu^2} * P_0 \\ \vdots \\ P_i = \frac{\lambda^i}{i!\mu^i} * P_0 \\ \vdots \end{cases} \tag{2}$$

Since $P_0 + P_1 + P_2 + \ldots + P_{\text{BOUND-1}} + P_{\text{BOUND}} = 1$, we have $\sum_{i=0}^{BOUND} P_i = 1$, and thus

$P_0 = \frac{1}{1+\sum_{i=1}^{BOUND} \frac{\lambda^i}{i!\mu^i}}$ and $P_i = \frac{\frac{\lambda^i}{i!\mu^i}}{1+\sum_{j=1}^{BOUND} \frac{\lambda^j}{j!\mu^j}}$, $i = 1, 2, 3, \ldots$, BOUND.

Since the arrival process is assumed as a Poisson process, we can apply the principle of PASTA (Poisson arrivals see time average). (Wolff 1982) Thus, the average number of arrivals in the service system L is equal to $\sum_{0}^{BOUND} i * Pi$. Then, we get

$$L = \frac{\sum_{j=1}^{BOUND} \frac{\lambda^j}{(j-1)!\mu^j}}{1+\sum_{j=1}^{BOUND} \frac{\lambda^j}{j!\mu^j}} \tag{3}$$

We assume that the time quantum is a fixed value *quantum*; the normal transmission time of packages generated in the time period of *quantum* is *packagetranstime*; the transmission delay time is random variable *delaytime*; the time of playing packages generated in the time period of *quantum* is *playtime*; and the lag time is *lagtime*.

Let us first consider the worst case with the number of arrivals in the service system equals to the BOUND. In this case, the corresponding VM is fully loaded and no more new arrivals will be directed to the VM. Thus at each iteration, the service time for each task is *quantum* and the non-service time of an arrival is (BOUND-1)* *quantum* which is the summation of the service time for the other arrivals.

At each iteration, as shown in the left hand side of equation (4), the time for processing and transmitting videos via the VM equals the service time *quantum* plus the maximal of the transmission time *packagetranstime* + *delaytime* and the non-service time (BOUND-1)* *quantum*. The right hand side of equation (4) is the time that the User plays packages generated in the time period of *quantum*. The equation (4) states that the processing and transmitting time should be less than the User playing time. To provide a good service quality, we have the constraint stated in equation (4). That is, a service system that satisfies the constraint stated in equation (4) provides the User a no-delay video service.

$$quantum + \max(packagetranstime + delaytime, (BOUND-1)*quantum) < playtime \quad (4)$$

As shown in equation (4), the service quality perceived by each User is related with its corresponding VM's CPU throughput and the occupied bandwidth for each arrival. If *packagetranstime* + *delaytime* > (BOUND-1)* *quantum*, the time of transmitting packages is longer than the non-service time of the arrival. It means that when the CPU starts to handle the task in next time quantum, the transmission of packages generated the previous time quantum is still executing. In this case, the bottleneck is the transmission speed. On the other hand, if *packagetranstime* + *delaytime* < (BOUND-1)* *quantum*, the time of transmitting packages is shorter than the non-service time of the arrival. It shows the bottleneck is the processing ability of CPU.

For the case that the service system cannot satisfy the constraint stated in equation (4), we define the lag time *lagtime* that measures the delay that the User may experience (at each iteration) while watching the videos. The larger *lagtime* is, the worse the service quality is.

$$lagtime = quantum + \max(packagetranstime + delaytime, (BOUND-1)*quantum) - playtime \quad (5)$$

Now let us consider the average case. Since the average number of arrivals in the service system will not exceed the BOUND, we have the equation (6). To meet this service quality, there is the constraint stated in equation (7).

$$L < \text{BOUND} \tag{6}$$

$$\textit{quantum} + \max(\textit{packagetranstime} + \textit{delaytime}, (L-1)*\textit{quantum}) < \textit{playtime} \tag{7}$$

According to the environment, the value of parameters which is discussed above is different. The value of the time quantum *quantum* depends on the CPU, the normal transmission time *packagetranstime* and the transmission delay time *delaytime* depends on the process unit and bandwidth and the time of playing packages *playtime* depends on the computer processing ability of the User. If we have all the value of the parameter, we can calculate the proper value of BOUND we set by the constraint – equation (7). We can find out the lower bound which can satisfy the service quality and decide the load balancing strategy.

requiredtime is defined as the number of complete quantum periods required for the service of an arrival. An arrival will exit during the execution period following the *requiredtime*th feedback. From the definition of *requiredtime*, the value of *requiredtime* for a service time x is such that

$$k*\textit{playtime} < x < (k+1)\textit{playtime} \tag{8}$$

In other words, *requiredtime* takes on the value k for an arrival requiring k returns to the queue. This is true since waiting time is only the time spent in the queue, and does not include the service time.

The experience time that the User spent in the service system regarding the PHASE I is equal to $\frac{2d-\lambda d^2}{2(1-\lambda d)}$. The experience time that the User spent in the service system regarding the PHASE II equals *playtime* requiredtime*. *requiredtime* occurs according to the Poisson probability distribution.

And the total experience time of the User is the experience time in PHASE I plus the experience time in PHASE II, which is shown in equation (9).

$$\frac{2d-\lambda d^2}{2(1-\lambda d)} + E(\textit{requiredtime})*\textit{playtime} \tag{9}$$

After choosing the first video, the User spends the average experience time stated in equation (9). If the User doesn’t choose any other videos in the following service time and the constraint - equation (7) is satisfied, the User won’t waste any time to wait for videos. It will spent the process time *quantum* + max(*L*, w* *quantum*) + *playtime* to execute the process from to segment packages to deliver to the User and to play if the User choose the other videos (issues a new service request).

5. Evaluation

We have conducted a questionnaire survey on the iPalace Video Channel service. This experiment lasts fifteen days. To simplify the study, we only provide the video service via the DC Collection. In this experiment, we use the average of CPU load instead of the number of arrivals of the corresponding VM handles with as the control variable. We set the average of CPU load as the upper bound to 0.1. The minimal load VM is iteratively selected from the cluster as the candidate for the coming requests. The Dispatcher did not set to count the number of arrivals of the corresponding VM in this experiment.

In the following discussions, we only report the data from 2012/3/4 00:00 to 2012/3/5 14:12. We do not report the result of Worker 1 (W1) because W1 also works as the Dispatcher while W1 works to provide the video service to users.

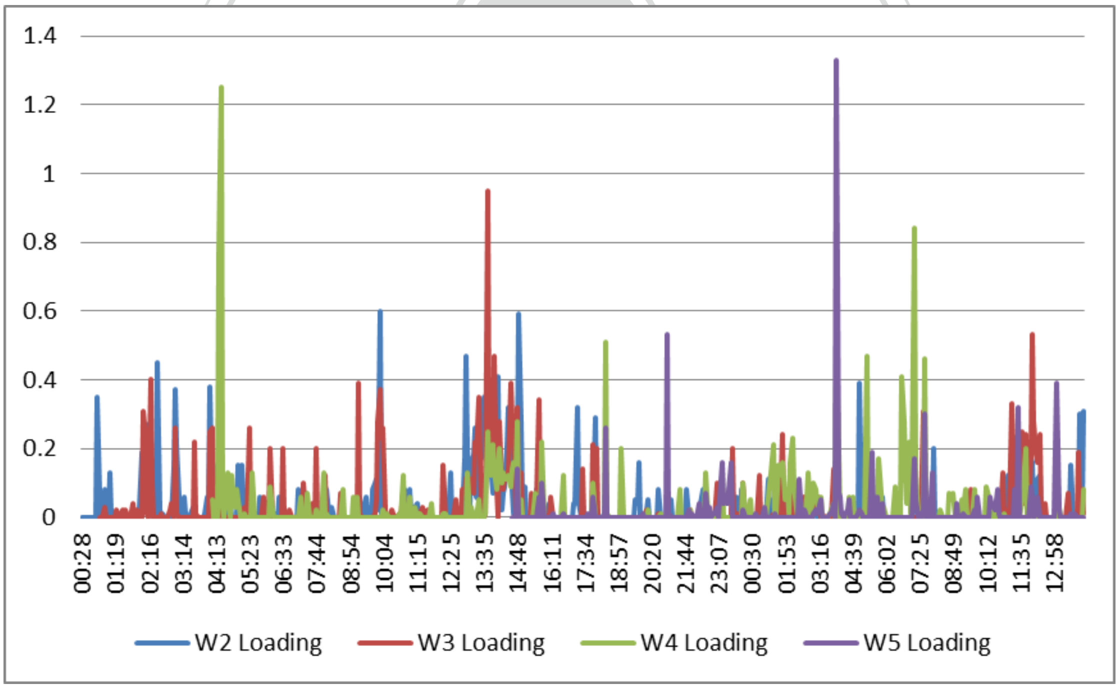

Figure 15: Change of the average of CPU load of VMs in the cluster. The X axis is time in minutes and the Y axis is the average of CPU load

Figure 15 shows the change of average of CPU load of VMs in the cluster. The X axis is *time* denoting the time epoch in minutes. The Y axis is the average of CPU load. A new VM is initiated when the minimal average of CPU load of running VMs in the cluster exceeds 0.1. At the beginning (time 00:00), only W1 in the cluster handles all the requests. It lasts 26 minutes and hits the average of CPU load to 0.26; then a new VM W2 (dark blue line) is invoked to share the workload. As shown in Figure 15, W3 (red line) is invoked at 00:49, W4 (green line) at 04:02 and W5 (purple line) at 14:34. Intervals between a new VM being invoked are 26 minutes, 23 minutes, 3 hours and 13 minutes, and 10 hours and 32 minutes. It is reasonable since that the more running VMs in the cluster, the more load can be taken and the longer the interval between a new VM being invoked.

Another interesting finding from Figure 15 is an observation of load balancing: that is, the peak load does not accumulate by a certain VM in the cluster. For instance, as shown in Figure 15, the peak load can be appeared at any one of W2, W3, W4 and W5. None of the peak load lasts on a specific VM for a long period of time since the new arrivals are dispatched periodically to a new VM (i.e., another running VM having a lower load or a new initiated VM). The load balance mechanism shows its effectiveness in this setting. However, the peak may be large enough to deteriorate the service quality. Thus, to guarantee an acceptable service quality, we integrate the load balance mechanism with the setting of an upper bound of number of arrivals to be served by a single VM.

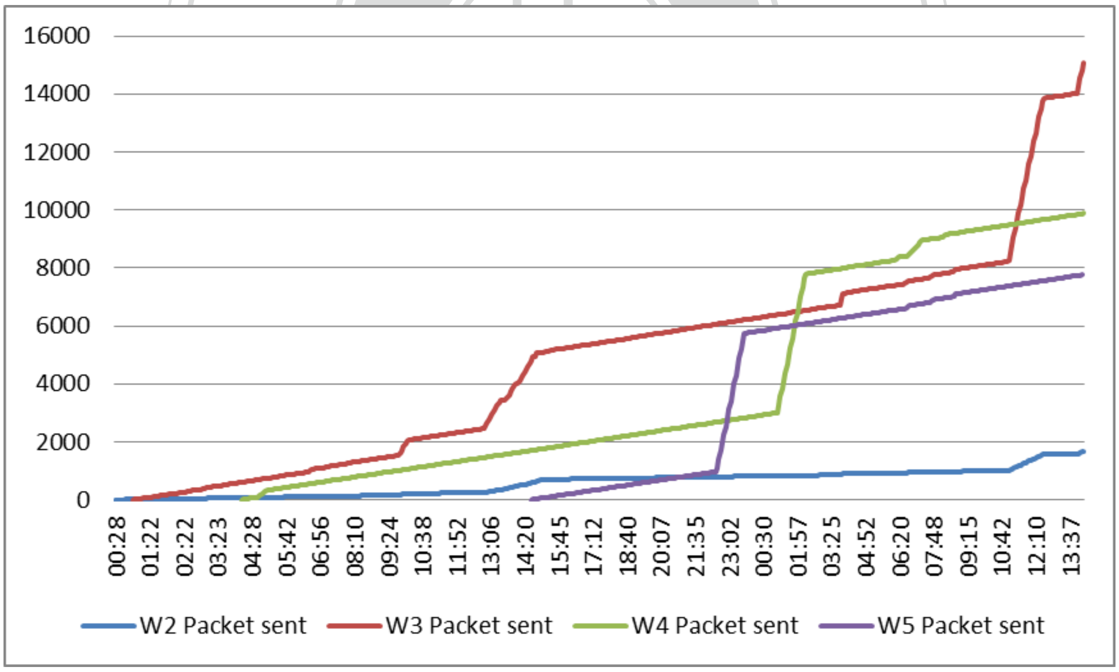

Figure 16: Change of packets sent in cluster VMs. The X axis is time and the Y axis is number of packets.

Figure 16 shows the accumulation of the number of sent packets of each VM in the cluster. The X axis is time and the Y axis is the number of packets. When the slope gets sharp, it means that the VM is sending a large amount of packets at that moment. One can observe that the VMs are taking turns of the increase, indicating the balance of the workload.

Figure 17 shows the change of requests per second of each VM in the cluster. The X axis is time and the Y axis is the value of HTTP requests per second. As shown in Figure 17, the value of HTTP requests per second gathered from Apache server is very low (<0.05) due to the nature of video-based service. Compared to the text-based webpage services, there are fewer click-and-responses among the server and the user in the video-based service.

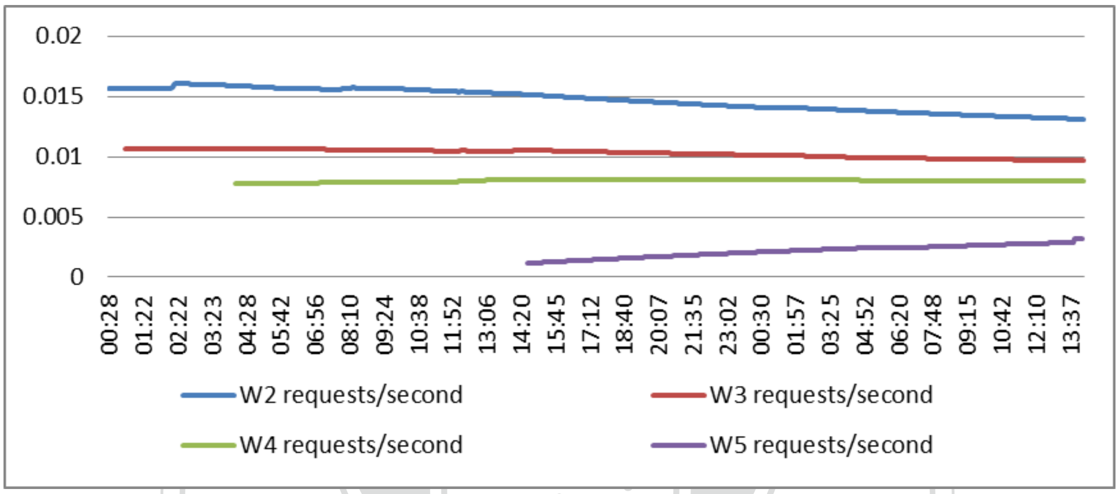

Figure 17: The requests per second in cluster VMs. The X axis is time and the Y axis is value of the requests/second.

Figure 18 shows the change of the number of established connections of each VM in the cluster. The X axis is time in hours and the Y axis is the number of established connections. We have approximate 3 to 8 connections of each VM in the cluster. As shown in Figure 18, the connections are distributed roughly in balance among the VMs in the cluster.

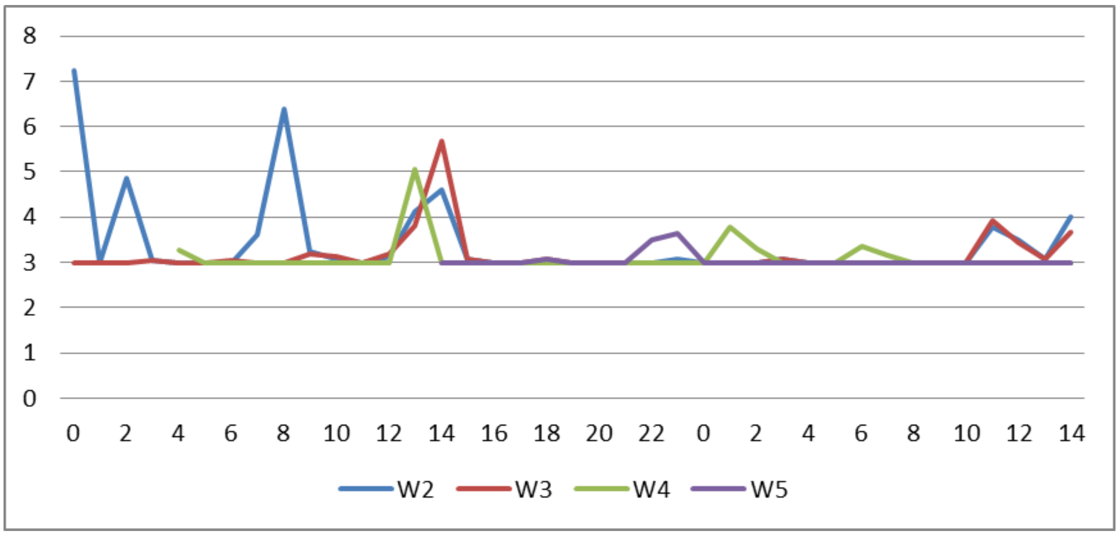

Figure 18: Number of Connections established. The X axis is time in hours and the Y axis is number of Connections established.

In sum, the preliminary result shows some promise of our cluster structure and load balancing algorithms. However, we find out some drawbacks from the result of the experiment. We distribute requests to the corresponding VM so that all the requests come within the period of DURATION will be assigned to the corresponding VM. If requests increase suddenly within the DURATION, it is possible the corresponding VM gets overloaded. Thus, we let the Dispatcher to count the number of arrivals the corresponding VM handing with. When the number of arrivals equals the BOUND – the upper bound of the number of arrivals that each VM can handle with, the Dispatcher will initiate a new VM, add it to the cluster, and update the corresponding VM stated in the internal XML file of the Monitor to select it as the corresponding VM.

Even though we distribute requests to VMs in balance, requests might increase suddenly in the DURATION which may violate the service quality. Because the iPalace Video Channel promise a high-quality and uninterrupted service, everything might violate the service quality is unacceptable. Thus, we design the two phase service system stated in section 4.

Another interesting finding is that the value of the request/second is much smaller compared to the result of the experiment conducted by Adler. Instead of video-based websites, Adler conducts an experiment which focuses on text-based website. It shows the difference of load balancing between video-based website and text-based website. There are a lot of requests in a short time for requesting content in text-based website. However, requests will not be sent again until users request another video in the video-based website. That is why the value of the request/second in video-based website is much smaller than in text-based website.

6. Discussion

In this paper, a theoretical study with the queuing model is conducted to understand the setting of video-based service provided through the cloud framework. By sampling the service in practice, we also conduct an online experiment for understanding the effect provided through a heuristic mechanism of load balancing.

The theoretical study shows that the service quality of the video-based service depends on the CPU throughout and the bandwidth of network regarding the cloud infrastructure. Intuitively, it comes to mind that the CPU throughout of the server and the client as well as the bandwidth of the network dominate the service quality. The Equation (4) in Section 4 (*quantum*+max(*packagetranstime*+*delaytime*, (BOUNG-1)* *quantum*) < *playtime*) confirms this relationship. It explicitly specifies how the CPU throughout and the bandwidth affect the service quality. Specifically, *quantum* (the time quantum) and *playtime* (the time of playing packages generated in a time quantum) both are determined based on the CPU throughout. And *packagetranstime* (the normal transmission time of packages generated in a time quantum) and *delaytime* (the transmission delay time) depend on the bandwidth. For different system and network environments, the constraint shows how we adjust the setting to provide smooth video services. If the values of *quantum*, *packagetranstime* and *delaytime* are larger (poor network transmission), we should set the BOUND smaller (so that each VM handles less connections) to avoid potential violation against the service quality. On the other hand, if *packagetranstime* is larger (a sequence can be displayed a long time on the client side), it is possible that we can set the BOUND larger to allow a VM dealing with more connections simultaneously. Another interesting finding is that theoretically, we can estimate the proper settings to satisfy the service quality given the performance of the computer and network environment (by observing systematical measurement), and hence can be applied to different system environment and architecture. The value of system reliability also can calculate from this equation.

In this paper, we propose that the backstage of the video service – the iPalace Video Channel is under the two phase service system architecture. The service system regarding the phase I is a multichannel, single-phase service system with a variant amount of VMs such that every arrival is served by a VM and there is no balking, and is responsible for receiving the user initial request and responding with the index webpage which directs the user to the corresponding VM. The service system regarding the phase I is a M/D/1: ∞/∞/FCFS queuing system and is designed to satisfy the service quality and accomplish the load balancing mechanism. The service system regarding the phase II is a M/M/1: ∞/BOUND/round-robin queuing system and is responsible for setting up the connection between the User and the corresponding VM and providing the subsequent video service. Defining the type of the service system helps us systematically formulate the relationship between the service quality and the CPU throughout of the server and the client as well as the bandwidth of the network.

The two phase service system is designed especially for video-based services because the traditional architecture of the text-based service system is not suitable to the video-based service. Enterprises who intend to provide video services could adopt our queuing model to control the service quality more validly. However, our load balancing strategy is applied to the datacenter in single location. In the case of datacenters in multi-location, enterprises could combine geographic load balancing strategy and our queuing model.

We present the load balance mechanism on the premise that satisfying the service quality. We set up a VM - the Monitor which periodically detects the number of arrivals served by each running VM, selects the VM that has the minimal number of arrivals as the corresponding VM and the next coming arrival(s) will be directed to this VM. On the other hand, if it exceeds the bound, we will increase the size of the cluster by initiating a new VM to join the cluster and dispatch the next arrival to this VM. In the end of monitoring procedure, the unemployed VMs will be released to an idle VM pool. Another VM – the Dispatcher dispatches the arrivals to the corresponding VM, and initiates a new VM and selects it as the corresponding VM when the number of arrivals that the corresponding VM handles with counted by the Dispatcher equals the BOUND.

We conduct an experiment by sampling the video-based service in practice and evaluating the presented approach online. In the experiment we observe that the peak load does not accumulate by a certain VM in the cluster and none of the peak load lasts on a specific VM for a long period of time. The load balance mechanism shows its effectiveness in this setting, but we have to integrate the load balance mechanism with the setting of an upper bound of number of arrivals to be served by a single VM to avoid the peak to be large enough to deteriorate the service quality. We also observe that there are fewer click-and-responses among the server and the user in the video-based service compared to the text-based webpage services.

Most of the researches of load balancing do not take the service quality into consideration, but focus on the performance of the load balance mechanism by conducting experiment or theoretically inferring. Except theoretically inferring and conducting experiment, we also apply queuing models into the load balancing mechanism and try to keep the service quality in high standard in this paper.

References

Adler, B. 2010. "Load balancing in the cloud: Tools, tips, and techniques," Technical report, RightScale, Inc..

Adhikari, V. K., Jain, S., Zhang, Z. 2010. "YouTube traffic dynamics and its interplay with a tier-1 ISP: An ISP perspective," *Proceedings of the 10th annual conference on Internet measurement*, pp. 431-443.

Amazon Web Service, available at http://aws.amazon.com/, accessed March 23rd, 2012.

Amazon ELB, available online at http://aws.amazon.com/elasticloadbalancing/, accessed March 23rd, 2012.

Aicache Web Accelerator, available online at http://aicache.com/, accessed March 23rd, 2012.

Adan, I., Resing, J. 2001. Queueing Theory. Eindhoven, The Netherlands, Department of Mathematics and Computing Science, Eindhoven University of Technology.

Bilderbeek R., Hertog P., and Marklund G. 1998. *Services in innovation: Knowledge intensive business services (KIBS) as co-producers of innovation,* 2000, STEP group, SI4S report, 3.

Cherkasova, L. 2000. "FLEX: Load Balancing and Management Strategy for Scalable Web Hosting Service," *IEEE Symposium on Computers and Communications*, pp. 8 - 13.

Christodorescu, M., Sailer, R., Schales, D., Sgandurra, D., Zamboni, D. 2009. "Cloud security is not (just) virtualization security: a short paper," *Proceedings of the 2009 ACM workshop on Cloud computing security*, pp. 97-102.

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., Newling, T. 2004. "Patterns: service-oriented architecture and web services," IBM Corporation, International Technical Support Organization.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. 1999. Berners-Lee, T.. "Hypertext transfer protocol – http/1.1", IETF RFC 2616.

Google AppEngine, available online at https://developers.google.com/appengine/?hl=zh-TW, accessed March 3rd, 2012.

HAProxy, available online at http://haproxy.1wt.eu/, accessed March 3rd, 2012.

Hoffman, D. L. and Novak, T. P. "Marketing in Hypermedia Computer-Mediated Environment Conceptual Foundations," *Journal of Marketing*, Vol. 60, 1996, pp. 50-68.

Ishii, A. and Suzumura T. 2011. "Elastic stream computing with clouds," *IEEE International Conference on Cloud Computing*, pages 195–202.

Johnson, S. P., Menor, L. J., Roth, A.V., and Chase, R. B. 2000. "A Critical Evaluation of the New Service Development Process," *New Service Development*, Thousand Oaks, Calif, pp.18.

Krishnan, R., Madhyastha, H. V., Srinivasan, S., Jain, S., Krishnamurthy, A., Anderson, T., Gao, J. 2009. "Moving Beyond End-to-End Path Information to Optimize CDN Performance," *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference IMC*, pp. 190-201.

Liu, H. and Wee, S. 2009. "Web Server Farm in the Cloud: Performance Evaluation and Dynamic Architecture," *Cloud Computing*, pp. 369-380.

Lipsky, L. R. 2009. "Queuing theory: A linear algebraic approach," *Springer Verlag*, pp. 28.

Microsoft Azure, available online at http://www.windowsazure.com/zh-tw/, accessed March 3rd, 2012.

Maglio, P.P. and Spohrer, J. 2008. "Fundamentals of service science," *Journal of the Academy of Marketing Science* (36:1), pp. 18-20.

Moreno-Vozmediano, R., Montero, R. S., Llorente, I. M. 2011. "Elastic management of web server clusters on distributed virtual infrastructures," *Concurrency and Computation: Practice and Experience* (23:13), pp. 1474-1490.

Namjoshi, J and Gupte, A. 2009. "Service Oriented Architecture for Cloud based Travel Reservation Software as a Service," *Cloud Computing*, pp. 147-150.

National Palace Museum, available online at http://tech2.npm.gov.tw/da/eng/active_news_1.html , accessed May 3rd, 2012.

Nobel , C. 2011. "A New Model for Business: The Museum," accessed at 12/28/2011 via http://hbswk.hbs.edu/item/6770.html.

Phifer, L. 2006. "Establishing a Geographically-Distributed Internet," Technical report, Coyote Point Systems, Inc., available at http://aws.amazon.com/, accessed March 23rd, 2012.

Piórkowski, A., Kempny, A., Hajduk, A., Strzelczyk, J. 2010. "Load Balancing for Heterogeneous Web Servers," *Communications in Computer and Information Science* vol. 79, pp. 189-198.

Patterson, D., Armbrust, M., Fox, A., Griffith, R., Jseph, A. D., Katz, R. H., Kownwinski, A., Lee, G., Rabkin, A., Stoica, I. 2009. "Above the clouds: A Berkeley view of cloud computing," Tech. Rep. No. UCB/EECS-2009-28. Berkeley, CA: University of California.

Rasch, P. J. 1970. "A queueing theory study of round-robin scheduling of time-shared computer systems," *Journal of the ACM (JACM)* (17:1), pp. 131-145.

Research Development and Evaluation Commission 2011, available online at http://www.rdec.gov.tw/mp110.htm, accessed March 23rd, 2012.

Silberschatz, A., Gagne, G., Galvin, P. B. (2004) "*Operating System Concept 7th edition,*" Addison-Wesley.

Sakata, M., Noguchi, S., Oizumi, J. 1971. "An analysis of the M/G/1 queue under round-robin scheduling," *Operations research*, pp. 371-385.

Turner, M., Budgen, D., Brereton, P. 2003. "Turning software into a service," *Computer*, (36: 10). pp. 38-44.

Vargo, S. L. and M. A. Akaka. 2009. "Service-dominant logic as a foundation for service science: clarifications," *Service Science*, (1:1). Pp. 32-41.

Wee, S. and Liu, H. 2010. "Client-side Load Balancer using Cloud," *Proceeding SAC '10 Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 399-405.

Wolff, R. W. 1982. "Poisson arrivals see time averages," *Operations research* (30:2), pp. 223-231.

Youtube, available online at http://www.youtube.com/, accessed March 23rd, 2012.