**Assignment Two for CS5223**

**Technique Paper Literature Survey**

|  |  |  |
| --- | --- | --- |
| **Topic of the survey:** | **Server Architecture** | |
| **Length of the survey in number of pages (excluding appendix)** | **Survey report: 15 pages**  **Appendix: 3** | |
|  | **Member #1** | **Member #2** |
| **Team Member’s matric number** | **A0159354X** | **A0091827X** |
| **Team Member’s name** | **Liu YuanCheng** | **Lu Shumin** |

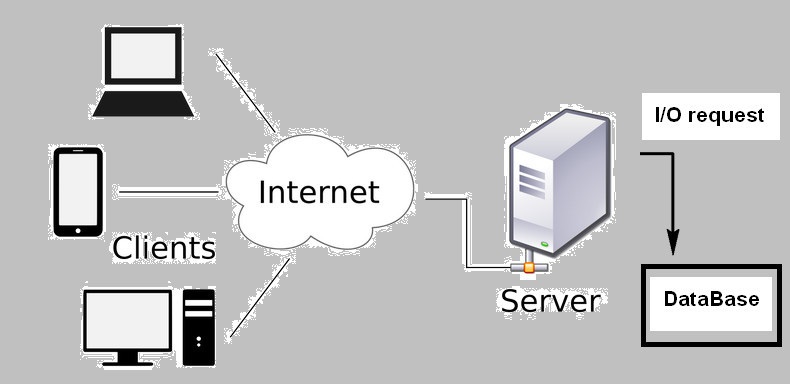
**Topic Introduction**

Introduction of server and client–server model architecture

The wide developed internet technology connected people together, for the organization / people / companies who want to provide service through the network, the servers are needed. Generally, a server is a set of computer programs which are running on a computer/ a distributed computers system to provide functionality to other programs or devices. In the current word, most of the server system are a set of computers with some distributed manage program. It is used as the central repository of data, resource and various programs that can share to the users in a network. There are two aspects need to be considered when doing the server technique development: hardware and software. In our research we will focus on the software aspect of the server architecture design.

The normal server software and program model is client–server model. The basic requirement of the client–server model architecture is a single overall computation is distributed across multiple processes or devices. A server can server can serve significate number of clients with in an acceptable response time, and one single client program can connect multiple servers for service. With the development of the distributed computer system, many technology and methodology are created to solve the problem when the server handle the users request and improve the performance of the system.

The viewable architecture of normal network server is very simple as shown below. It contents two parts: one part is a computer / several computer in front to handle the request from different clients and the other part is the computer system to do the computing work and data process (transition, exchange and storage) The architecture looks simple, but when people develop the system to handle many the clients, a lot of problems appeared and need to solve. As shown below, the server need to handle the request from several different devices and process the data I/O request in a user acceptable response time.

-----------------------------------------

The current server architecture in the real word is much complex than the diagram on the left. The file-wall unit, PSTN gateway, backup-recover unit and different kinds of sub-server system are working together as distributed system.

-----------------------------------------

Introduction of Server category and architecture design requirement

The view of a server system to its user/client is nearly about a single computer running some program, but inside the system may different kinds of sub-server system are working together as distributed system. The basic server architecture is simple, but the computer server can separate to several scenarios and each scenario has its own architecture design requirement:

Application server: This kind of server is used to host web apps (the program which run in the web browser to the client. The user can run and use this Apps’s function without install it in their computer). It should provide the advantage of event-driven programming to handle the user continuously requirement in an acceptable response time.

Catalog server: It provide an index list of contents of information across a large distributed network. It is used for user to find the something in the website or on the network. The main requirement of this kind of server the fast I/O data process.

Communication server: It is used to handle the large amount of the user’s network request and the data flow/ function call inside the distributed server system. Such that the multi-threading process and the ability to handle heavy work intensity is required for this kind of server.

Computing server: Run the main program to handle user request, control and manage other different kind of servers to finish the tasks. Fast computing speed and program running efficient is the base demands. For this kind of server, a flexible design, re-useable and extendable functionality are required by the server developer for adding new function, debugging and doing maintenance.

Database/File server: These kinds of server are used to handle the I/O, message filter, information save/load problems to help the computing server to finish the jobs. So high efficiency data exchange and data synchronization between different computer are urgently required.

There are every other kind of server such as mail server, media server, sound server, proxy serve rand Web server which share one or server aspect and design requirement with the above servers. In this paper we will pay some attention on the gamer server architecture and share some experiment about the Maze server design during the assignment1 in the next section. The game server is a combination of the server shown above, the multi-player game server need to handle massive concurrency client requirement and keep game data synchronization for all the players.

Introduction of the technique Paper view:

We reviewed 5 papers about the design of server architecture and 4 main kinds of server architecture are introduced and one comparison discussion about how to improve the server performance is also listed. We briefly do an introduction about what the five papers did:

“SEDA: An Architecture for Well-Conditioned, Scalable Internet Services”

* This paper provides an improvement architecture of the event-driven server. It uses explicit queue method and dynamic resource controllers to split the jobs to different stage to balance the whole workload, so the system can perform higher when support massive concurrency demands compared with normal event-driven server under the same hardware condition.

“Capriccio: Scalable Threads for Internet Services”

* Capriccio is a scalable use-level thread package for improving the thread based server to handle. This server architecture is aiming to improve the thread based server system to provide similar or even higher performance as the event base server. For the server data exchange, it introduces the new method “linked stack management” and “resource-aware scheduling” to minimizes the amount of wasted stack space, make data stack grow or shrink at runtime to handle the data assessment and provide a better scheduling and admission control to adapt to the system’s current resource usage.

“Web Prophet: Automating Performance Prediction for Web Services”

* Web Prophet is focus on the Web server design, it employs a novel technique to analyses the dependency of the object in the webpages. Then it automates performance prediction for web services to optimize the loading sequence of a web pages to the client to highly decrease the page loading time and page error rate.

“Events Can Make Sense”

* This paper introduces another improved event-base server design for managing the concurrency client request for the application server. The program developed with Tame can avoid some “stack-ripping” problem for normal event-driven server, it also reduces the runtime overhead and improve the program’s compatible for different server platform.

“Why Events Are A Bad Idea (for high-concurrency servers)”

* This paper provides a discussion and experiment about the comparison between event-base server and thread-base server architecture when create a highly concurrent application. Base on the experiment data, the paper wants to show that a good implement threads-base server can achieve all the strengths of events-server, including support for high concurrency, low overhead, and a simple concurrency model.

After paper reading and research, we found the main technique discussion and argument are between base-driven and the threads-base server. At the pure programming, mathematical and methodology level, people made a lot of development to make one architecture take the advantage of the other and avoid the disadvantage of the method itself. At the industry and practical usage level people are preferring to combine them together to do the server design as each of the two-kind architecture cannot fully replace the other. We think there is not a kind of “Best” architecture for the server design, for different kind of server as shown in the previous page, the “selection” needs to depend on several factors. (here needs about 1/2 graph to introduce what factor)

**Problem specification of server architecture**

As introduced in the previous section clients’ requests are very different amount different kind of server, which made the problem for server architecture designer are also different. But normally there are several common problems for most of the server design. The five papers discuss some common, but detail of the problem which designer need to face:

Concurrency service request handling problem.

The most frontier problem when a server is connected to internet is about how to process the massive concurrency demands/request. Some maturity in math and algorithms are urgent need for the request service queue to avoid the request jamming or queue’s buffer overflow.

Sub-System coordination and control problem.

The Server is a distributed system, it looks as a single computer to the outside, but inside how to arrange the working of different subsystem and control the data flow is a very complex problem for the designer to consider.

Server data management and exchange problem.

Even a very good arrangement/sub-system management system is designed, without support of good data exchange and resource usage architecture the system still cannot match our server design goal. The problem of how make single computer file save/load, memory access efficiency and how to make multiple computers’ data synchronize is as important as the server control program.

Server resource utilization problem

Every computer which made up the distributed server system has its hardware limitation. For a server designer, he cannot avoid considering the problem of making the server program running efficiently, increasing the resource utilization and decrease the server overhead. At the same time, not all computer can run all the time, the server backup, system recovery, service fault tolerance and the aspect of server maintenance/debug/whether it is easy for function extension and further development are all very import problem need to be solved.

Server response time and delay problem.

The client’s request go through the server and the response is sent to the user. The server response time is also a very critical problem. The service need to control their response time under an acceptable time interval for the user and make the service delay as short as possible. This problem is very critical for the Web server or the game server.

For the methodology and technology introduced in the 5 papers the SEDA and Capriccio focus on the problem of concurrency request handling. Capriccio and Tame provide some great solution to increase the unification of the system resource and increase the data exchange speed. “Why Events Are a Bad Idea” give a valuable comparison about the performance of event-driven base server and thread based server when facing the concurrency request handling problem. The Web Prophet provide a flexible solution for decreasing the page load times problem.

**Technical Introduction and logic summery of the papers**

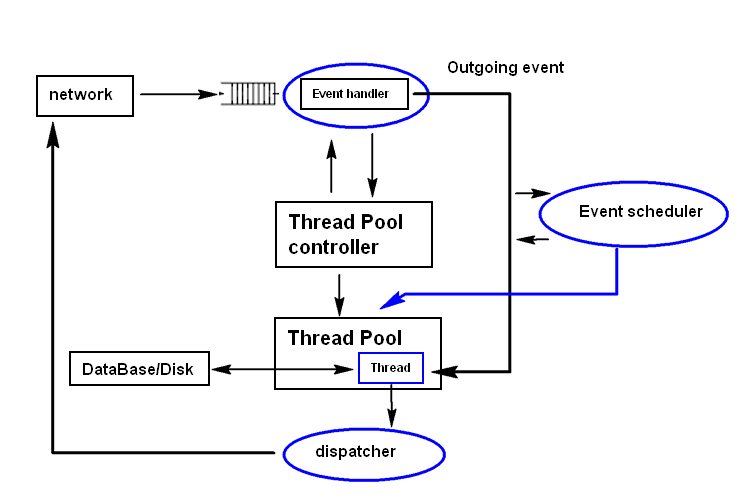
Main view of the contents of the papers

Different server has their one special architecture and functions with different methodology. But the main overview of the process flow of them are nearly same. In this literature survey paper, we did our research based on 5 papers which mainly introduce 4 kind of server architecture which can be used or considered as a kind of technical solution for solving the problem when people are developing different kind of servers.

For all kinds of the server, the first step of the server-client request handling process is receiving the user requests which are massive concurrency demands. When the server system is handling the huge number of the request, there are 2 main Service-level agreements from the user:

The fist one is the server need to provide robust performance with a wide range of services. The second is the server need to provide acceptable response time of the service event under huge variations in load. Some client request handling program running on the server system use the thread base program modules and some are using the stage based program. The paper "SEDA: An architecture for well-conditioned, scalable internet services" introduced a kind of server architecture which combines aspects of threads and event-based programming models to manage the concurrency, I/O, scheduling, and resource management needs of Internet services. It follow the main design of the event-driven architecture (Event driven requirement handling) and use the thread based methodology as the basic programming for the detail computing process. So the response time for the request is fast and with multi-threading it can process large amount of concurrency demands at the same time. Its thread pool architecture makes the server hardware utilization efficient and scalable for the huge variations in load which may appear.

Based on the main steps introduced in the paper the work flow sequence of the staged event-driven architecture (SEDA) is this:



(I will add step index in the above flow diagram to introduce each step do what, that will take about ½~1 pages)

As shown in the above, when the client request get the server queue, the event handler will check with the thread pool control whether the server system(a set of distributed computers ) has “free” thread to handle the event, if the thread pool utilization is full the thread pool controller will create new thread for the event and response the event handler, then the event goes out from the event handler, it knows which thread it show go for process, then the event scheduler will scan the work intensity of the event, if the workload of the event is heavy, the event scheduler will ask the thread pool to create special I/O thread to help the main event thread to handle the data flow problem to improve the process time during the event handling stage. Then the dispatcher will dispatch the request result to the user according to the different stage. (Such as the user’s request has 3 separate unrelated question, when the thread finished process the first question, the dispatcher will provide the answer of the first question to the user fist then wait the thread to finish the others, so the response time of the server is increased)

The SEDA is good but not all the client request is complex and need so much I/O process of the access to the data base, if we use the SEDA to handle these kind of request, the stage creating time and the computing time of the dispatcher/scheduler is even longer than the time of the computing time. So other pure threading server is introduced in the paper “Capriccio: scalable threads of the internet service”. This kind of server architecture is used to handle a large amount client request which use light files access or assess the same resource in the server computer system.

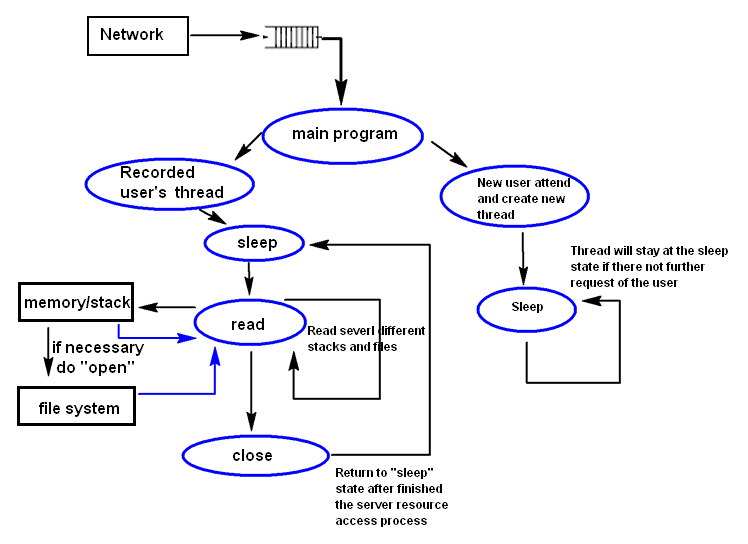
As most of the service is accessing the same resource/files in the system, these files will be pre-loaded in the computer’s memory (stack) to increase the resource time. So the programming of the Capriccio contents serial kernel level(memory/stack ) asynchronous I/O mechanisms.

As a thread base approach, Capriccio use loop mechanisms to handle a set of request to increase the total performance. From the experiment data provide from the paper, it shows the User-level thread can greatly reduce the overhead of thread synchronization.

To overcome the disadvantage of the loop design, it also provides some architecture to manipulate the advantage of the event-driven approach: Thread Scalability. It has a special architecture to monitor the incoming workload in the service queue. According to the paper’s introduction, it can “Improve the number of the thread according to the current require work load. Producers put empty messages into a shared buffer to pre-create the space for the workload increment.”

This kind of server is good to act as a network game server, such as a lot of players are attacking a boss in the game, each player send the “attack” request to the server and the data of the boss in the game(such as the hp, mp and the target player of the boss) should synchronous to every players. The players’ attack request contents little information but the number of the data is huge during every second. So the Capriccio will save the information of the Boss in the memory stack and create a thread for every player who attend in the attacking area. If the user does some action (attack/add debuf/ recovery other players) the server will access the data of the bass from the memory directly. If the user didn’t do any thing during one scan loop, its thread will “sleep” to save the server’s computation power and when the players do some action, their thread will “wake up” and the do the computer data changing and file reading thins. If suddenly many user has attend in the area or the user create a lot of request, the controller will increase the thread number to deal with the heavy workload for the next server loops

We use the above example to show process flow of the Capriccio:



(I will add step index in the above flow diagram to introduce each step do what, that will take about ½~1 pages)

Another kind of very important server is the web server. After analysis thousands of concurrencies request, we point our focus on how to handle one request fast and efficient. This is the key to match most of normal users’ service-level (response delay) arrangement. Web-Prophet makes the webserver’s performance more intelligent by using its key methodology: Prediction of the work load and Automating process adjustment.

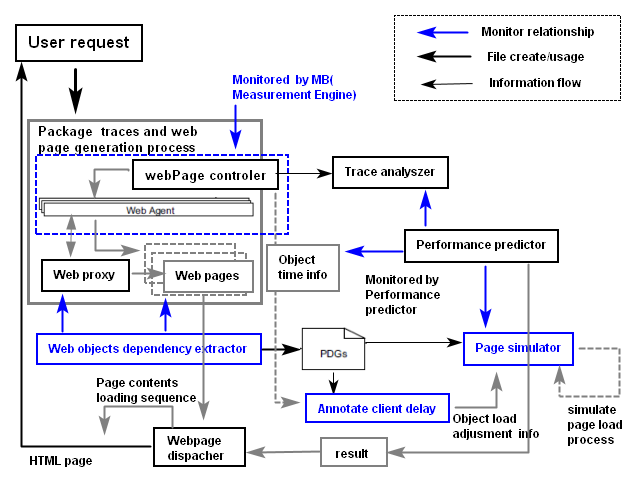
This kind of server is used to provide the HTML web pages to the client when the client type in the URL or click a link. There are 4 main part need to load inside the users browser:

* wireframe structure(the main HTML text document )
* Function (such as the javascript, flash plug-in and java-applet handle the user’ request)
* Self-explanatory content: some document, image or media content.
* Aesthetics: mostly refer to the CSS of the webpage.

For most of the normal user, their first consideration about a Web is its page load times. Poor page load times (PLT) result in low service usage, which in turn may undermine service income and there are 4 main time element which will make obvious influence of the server’s average page loading time:

* DNS lookup time: This is time to match the DNS in the user’s typed in URL to the IP address of the web server(the mount of time it takes a domain lookup to occur while a browser retrieves a resource.) for our web server architecture design we don’t need to consider about this.
* Network round trip time (RTT) : **round-trip delay time** (**RTD**) or **round-trip time** (**RTT**) is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received.(from wiki)This is also not the main aspect for the paper to consider for the server architecture design.
* server response time: there are several factors which can effect the server response time such as Databases, Traffic and Hosting, Resource Usage and Server Software. In the paper ” Automating Performance Prediction for Web Services ”, it provide a kind of server architecture to handle these factor and do the improvement.
* Client execution time: The Web applications rely on JavaScript or some other client-side scripting will take some time for function handling.

The main working flow of the Web-Prophet is show below:



There are three main steps in the Web Prophet’s working sequence:

1. Create the HTML pages base on the user’s URL request.
2. During the first page creation, use the “Measurement engine” to web network traffic, file I/O, distributed server computer can provide the information to the analyzer.
3. The Dependency extractor uses novel algorithm to infer dependencies between web objects. (The most dominate dependency factor is the download times of individual objects)
4. The performance predictor Implements a simple accurate method to simulate the page load process of a web browser based on the given dependency graph of a webpage.
5. The webpage dispatcher will come a load sequence of the page and the similar pages’ contents to the user base on the dependency graph.

According to the experiment result of the paper, the page load latency are decreased a lot when using the architecture design of Automating Performance Prediction.

< Events Are A Bad Idea (for high-concurrency servers) >

<A system for managing con-currency in network applications that combines the flexibility and performance of events with the programmability of threads >

Pros and Cons (Comments)

What are the nice properties of these techniques?

Event base server: Can handle the client request fast and accurate.

In the event-based server analogy, multiple customers might be served by a single shop employee – let's call him Bob. Bob delegates various steps that might take a while (like "find me item*x*in the back room") to other store employees. When Bob asks a helper for help, the helper scurries off to somewhere else in the store, and Bob can move on to help other customers while the original customer waits around for the helper to come back to Bob. When the helper does return, having finished their task, they will wait for Bob to come to a good stopping point with Bob's current customer, and then Bob can talk to the helper and original customer again.

Thread based server: Increase the utilization of the computer resource.

In the thread-based server analogy, each customer is served by their own shop employee. When the customer leaves, the shop employee can help another customer. The number of employees that can be helped simultaneously is directly tied to the number of employees at the store.

Stage base server:

A combination of the event-driven server and thread based serve, it use the event-driven methodology to handle (do the arrangement work) of the client’s request and use the thread based methodology to do the detail computing and servicing work of the request.

Scalable thread server and Automating Performance Prediction for Web Services:

These kinds of server are focused on increase the server’s utilization and handle the different kind work intensity at different condition. These kinds of computing power changeable methodology are very helpful for the distributed server system, so the computer’s computing power is fully used and it can also avoid most of the server crash situation.

What are potentially problems?

Event base server:

In an "event driven" model, if a handler will take a very long time to finish (i.e. by having a computationally intensive for loop inside), no other request will be handled during that time, because the event loop will not invoke the next handler before the current one completes. That's usually not an issue because of the asynchronous nature of Javascript.()

(use the game server example create in the introduction to replace the javascript example)

Thread base server:

In the "thread driven" model, if the handler takes a lot of time to complete, it won't hurt other threads much, because they can run at the same time independently.

Unfortunately, creating a new thread adds some overhead and if you need to handle thousands of concurrent connections, it might become a burden. That's why Node.js is considered fast - no matter how many connections you handle, there's only one thread [1](http://strongloop.com/strongblog/node-js-is-faster-than-java/). You just need to be a bit careful not to block in any of the handlers to keep things moving. Fortunately most of the time it's not that easy to write blocking JavaScript code.

(develop this java cript example to show the problem in own word)

Stage base server:

It will create a kind of “waste” of the computing power is the system can handle the request extremely fast.( The stage arrange program time is much longer than the computing time and request service time)

Control aspect: The server's working control flow is not viewable for the user/developer, so the maintenance and debugging process during usage is hard.

Programming aspect: The function/method caller and the callee sometimes are in different program/lib, so for understanding the whole logic for the process, people need to collect/restore all the live state.

(80% same as introduced in the paper, need to change late)

How do the technique compare with each other?

Is the technique revolutionarily novel or is it just some minor tweaks on existing techniques?

Are the techniques practically feasible?

Do they make unrealistic assumptions?

Some of the paper made unrealistic assumptions about the