SANTA CLARA UNIVERSITY

Load Balancing Algorithm and Architecture

Project of COEN296

Chris Yu

Kenny He

Sonny Gandhi

9/2/2013

In this paper, we will be discussing some faults with existing load balancing algorithms and architectures. We will propose a new load balancing algorithm and a new grid architecture that will utilize the new load balancing algorithm. The performance of the new load balancing algorithm will be compared with an existing load balancing algorithm.

Table of Contents

Preface

Many of the load-balancing algorithms out there have not kept up with the changing network environment, and so their goals/priorities may not be ideal for today's environments. In this paper, we will be introducing a new grid/cluster hybrid environment that can perform multilevel load balancing (if needed) in an attempt to improve response time for jobs submitted.

Acknowledgements

We would like to thank Dr. Ming-Hwa Wang for granting us with the opportunity to work together on this project and for providing us with an environment to showcase our work and findings with our peers and classmates.

Abstract

In this paper, we will be discussing some faults with existing load balancing algorithms and architectures. We will propose a new load balancing algorithm and a new grid architecture that will utilize the new load balancing algorithm. The performance of the new load balancing algorithm will be compared with an existing load balancing algorithm.

Introduction

Objective

We need to design a new algorithm/structure for load balancing to meet the new goals and adapt to the dynamic network environment. Also, if time permits, tackle the scalability and serviceability issues. We have implemented our architecture and will discuss its results.

Add more goals

1. The workload on each server of the cluster is under a "safe" level

2. Adapt to the new grid/cluster architecture (with scalability -- not limited by the count of servers; serviceability -- all servers, coordinators, central directory servers backup for each other).

3. The distance between the server and client is within a certain level to make sure that the network propagation and response delay is within a rational level (e.g. for VoIP: If the distance between the client and server is too long, the voice delay is intolerable.)

4. Quick response: The server selected may not be the best one to balance the workload, but the response time is the shortest so that a client need not wait for too long to be assigned a server.

5. Asymmetrical hierarchy, high scalability, with multiple stages of load balancing.

What is the problem

1. Dynamic change in current networks is not accounted for in many algorithms

2. More complicated network architecture (as I mentioned in goal 1 above)

3. There are new goals in the load balancing field

4. Old algorithms have not kept up with new goals or environments

Why this project is related to the class

This project involving a working knowledge of cluster and grid networks, as discussed in class. Also it deals with load-balancing algorithms, which are a important component of network computing. The project will utilize the advantages of various network concepts within the design. It will also touch on numerous concepts like reliability, availability, efficiency, and quality of service. In summary, the project will cover most of the major core concept behind networks.

Why the other approaches are not good

1. Many load balancing algorithms are not scalable with many nodes.

2. Other approach choose to load balance amongst all peers, this is not necessary because all we need to ensure is that the load at each server is at a safe/secure level (no need to balance while servers are safe/server)

3. Many algorithms do not consider the "log out" case and assume all computation nodes stay in the system.

Why we think our approach is better

1. Suitable for the more complicated cluster architecture in the real world,

2. has good scalability; RAS (refer to the "related to the class", part 4)

3. No single-point of failure

4. Balancing occurs only when needed (less time wasted doing unnecessary balancing)

5. Multi-stage balancing/hierarchy

6. Dynamic system architecture, to more closely simulate real-life systems that have servers die/drop out/log out

7. We will try to alleviate frequency information exchange when load balancing

Statement of the problem

To create a dynamic multi staged load balancing architecture and algorithm, to yield a lower response time for jobs submitted.

Scope of investigation

1. Recognize the problems

2. Design new cluster architecture

Needs to have: Servers, coordinators, central directory, and clients; network topology or connections, and the backup relationship; heartbeat between the servers and coordinators, coordinators and central servers for guaranteeing the server availability

3. Setup the new load balancing goals:

- Ensure workload of each server is at safe level.
- The distance between the client and server could be the shortest.
- Have an acceptable job response time.

4. Design the new algorithm:

The algorithm is open to future changes: We use strategy patterns to create functions for the goals evaluation. Whenever there are new goals, it is easy to create new strategy functions and hook it to our algorithm. We are combining different methods/algorithms together to create a better algorithm for our architecture.

5. Design a simulation program to simulate a big cluster, the servers, coordinators, workload request, server available or shut down, and the request came and dispatched. We will apply an existing algorithm and our algorithm on it and compare for result and evaluate the RAS (reliability, availability, and serviceability).

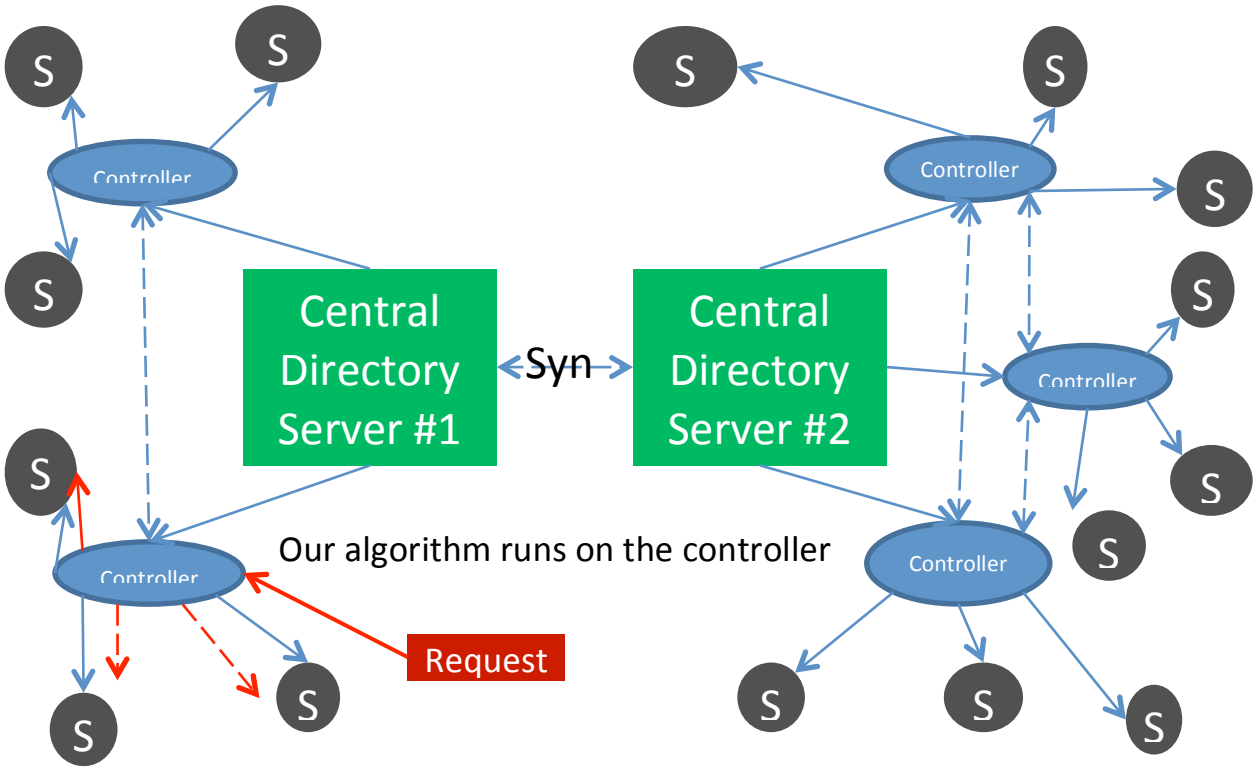
Theoretical Bases and Literature Review

Definition of the problem

Most algorithms dealing with grid or cluster networks did not account for dynamic changes within the current network. To solve this issue we are proposing a modified network architecture design that would be easier to scale and account for dynamic change. See diagram 1. In the diagram you can see there are:

- Two Central Directory Server that backup to one another
- A sub level network of controllers that control the flow of server load allocation
- Servers, or peer nodes, that compute job requests as received by its assigned controller

Figure-1: The network architecture



We will implement a new algorithm that a controller will use to determine load-balancing among the servers and also with its backup controller. With this new design, our goal of increased scalability and efficient response time will hopefully be achieved.

Theoretical background of the problem

A computer cluster consists of a set of loosely connected or tightly connected computers that work together so that in many respects they can be viewed as a single system. The components of the cluster are usually connected by LAN (local area network) with each node (server) running its own instance of operating system. Typically clusters are used to achieve higher computational speed and availability over that of a single computer. Usually the workload on a cluster consists of jobs that require great amount of parallel processing.

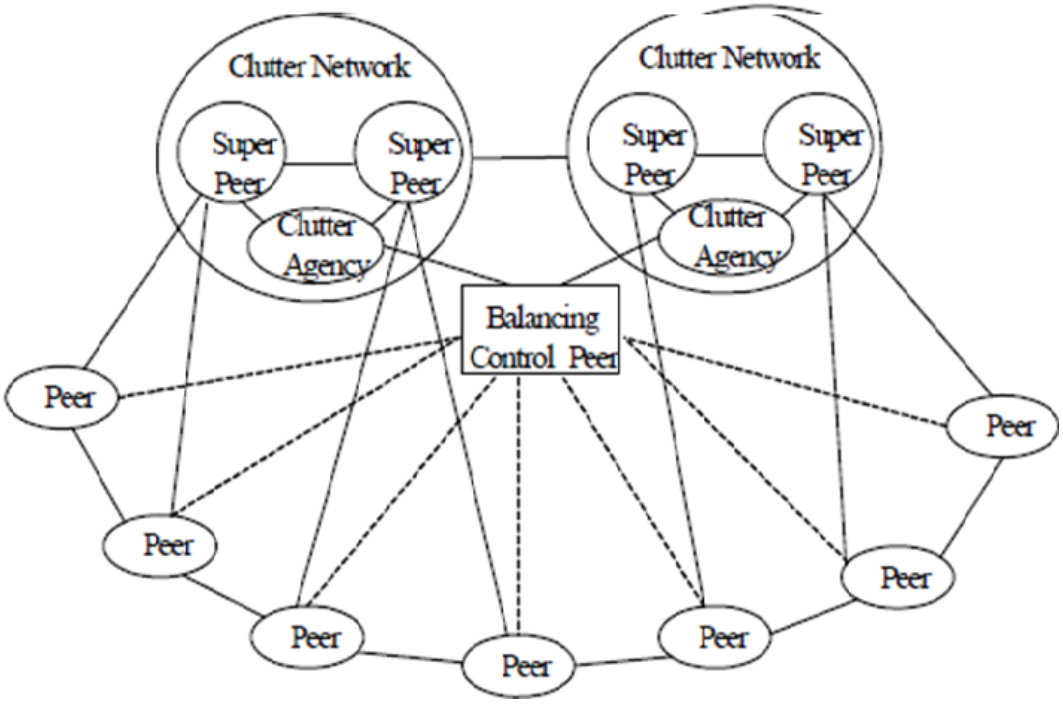
Grid is defined as a large collection as machines connected by a private network and offers a set of services to users. It acts as a sort of supercomputer by sharing processing power across the machines. Grid computing generally is a decentralized model where the computation could occur over many administrative domains. Typically a grid would be distributed geographically, sharing computer power to solve a single problem. Compared to cluster, it is significantly more spread out location wise, and operates on a much larger scale.

Load balancing seeks to improve the performance of a distributed system, usually in terms of optimal resource utilization, maximize throughput, minimize response time, and avoid overload, that done by allocating workload amongst a set of cooperating hosts and keeping processor idle time and inter-processor communication as low as possible.

Related research to solve the problem

The paper “A Load Balance Algorithm for Hybrid P2P Network Model” by Fu Xiao-ling and Xu Ying introduced a load balancing algorithm based on the following hybrid P2P network architecture:

Figure-2: The hybrid P2P network architecture in the existing paper

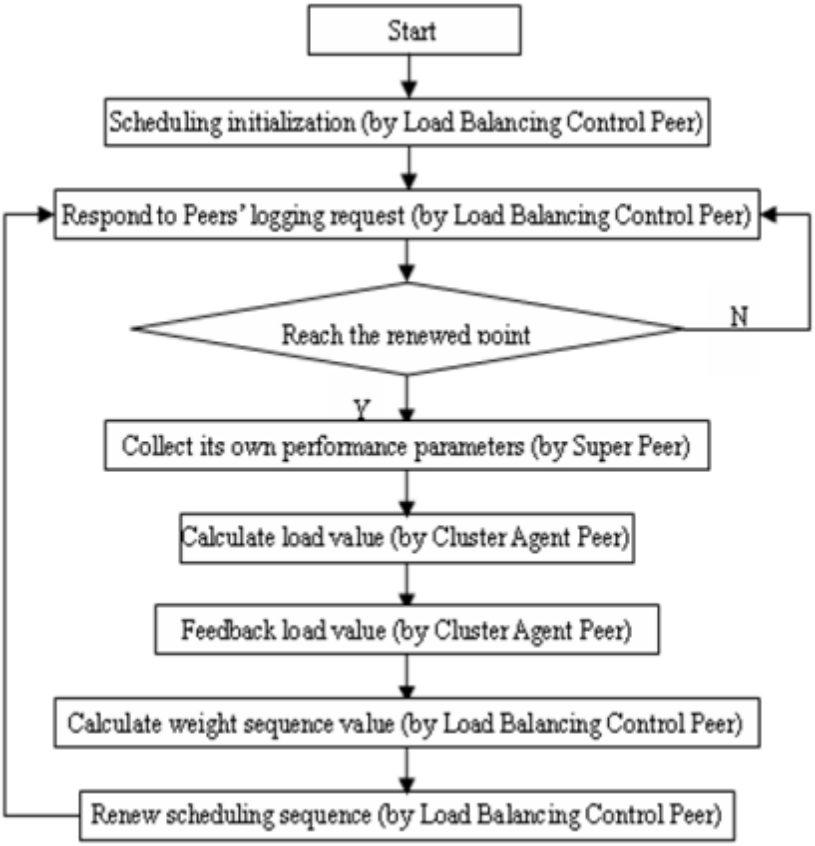


The paper presents a dynamic feedback adaptive scheduling algorithm to deal with the load balance of Super Peers in hybrid P2P network. The algorithm applies the adaptive peers' distribution login request by weighted rotation, and adjusts the Super Peers' scheduling sequence dynamically with genetic algorithms to make the Super Peers load in system tending to a best balanced state.

The characteristic of the algorithm (refer to Figure-3 in the next page):

- Characteristics
- Super Peers to handle the load
- Each peer has a weight
- Balancing Control Peer to collect the load weight of each login peer on the Super Peers and balance the load of them.
- When a new peer join the network, needs to round check the weight with the Super Peers to get best balance.

Figure-3: The flow chart of the algorithm



Advantage/disadvantage of those research

Good: Algorithm can account for nodes joining the network and load-balancing based on feedback from controlling peer. This allows for good allocation of workload.

Bad: Many algorithms did not consider the log-out case for when computation nodes leave the network. The algorithm was also susceptible to failures because they had 1-point of failure nodes. The architecture also is not very efficient in terms of scalability; the design will only work with limited number of nodes. There is no way to account for massive influx in nodes, and also can't support dynamic node entry and exiting.

Our solution to these problems

We will implement a network design as specified above in the definition of the problem. The algorithm at controller level will assign jobs to a server until it is at a predetermined capacity

level. If a server is full, it will load balance to a different server within the controller's region. If in the case that all servers are full, controller will flag the job and direct it to it's dedicated backup controller, and the backup controller will then assign and load-balance if need be. If both controller regions are full, then the job is sent back to CDS and it will redirect it to another controller.

Where your solution different from others

Our network design has more accountability for failures, as our CDS has a backup CDS, and our controllers have an assigned backup controller as well. With this design, we eliminate single point of failure, and increase availability. Another difference in our design is the constant updating of the servers, the server nodes will always send a pulse or signal of information to the CDS, thus informing CDS of every server's load and availability. With this feature, the decisions CDS makes will always be current and response time should be minimized.

Why your solution is better

We are considering additional goals and new environment that the existing algorithms never considered. This should hopefully result in a better algorithm for load balancing.

- Reduction of single point of failure
- Accountability for failure cases
- "Pulse" updating that optimizes response and decision time of the CDS
- load-balancing algorithm will be utilized less frequently, reducing computation time for the algorithm

Hypothesis or goals

Positive/negative hypothesis

- We will try to make a more stable and “safe” dynamically load-balancing architecture.
- We will try to make our system deliver a lower average job-response time than the existing algorithms.
- With our architecture, we will try to simulate both our algorithm and an established algorithm to compare response times and hopefully show our algorithm is better.

Methodology

How to generate/collect input data

We will do a simulation to compare the algorithms. Before the simulation, we will write a program to RANDOMLY generate the input data to simulate the real cluster architecture and the incoming requests:

1. Cluster architecture:

For each server, coordinator, central directory server: position, distance (not direct distance: = the distance * (a random number ranges between 1.05 and 2), role (totally two central directory servers backup for each other; even number of coordinators which can backup some of its neighbors when they are down), the neighbor relationship.

2. The incoming requests: A request is initialized by a client.

The position, workload, connected coordinator and the distance to it, etc.

3. The up and down of the server and coordinator.

Algorithm design

1. Implement an existing algorithm (in our paper, we can choose one) based on the cluster architecture, ignore the goals not considered;

2. Implement a new algorithm based on the cluster architecture and try to meet all the goals;

4. Implement the simulation application, and apply the existing and new algorithm in the simulations.

5. Run the simulation application, emulate the real cluster and requests (from the generated data), and keep record of the indexes (waiting time, processing time) of each request and the overall average index after simulating for a certain period.

6. Compare the indexes and give remarks to the new algorithm.

Programming Language & Tools

Algorithms and simulation application: Java

IDE: Eclipse, JDK 1.7

Input data preparation, output data analysis: Microsoft Excel with Visual Basic for Applications.

How to test against hypothesis

We will run the program with the generated input data and compare total run-time between the two algorithms. We'll analyze the speed differences of the two algorithms and whether Algorithm 2's speed improvement is significant enough to warrant the hypothesis. Also, we will look at the failure rate of the requests of each algorithm and analyze the significance of that difference.

Implementation

In this section, we will discuss each part of the code we created for this project. We will then show a design layout chart that shows how the classes relate to one another. Lastly, we will discuss 2 flowcharts that describe both the benchmark algorithm, and the algorithm that we created to conduct the load balancing within our architecture.

Code

There are many classes and interfaces defined in our project; this section will briefly describe each of them. To see how they have been implemented, please see our source code in the appendix.

The Controller in our architecture is the manager for an individual cluster of servers, it is in charge of many servers, and keeps track of their loads to communicate with the CDS.

The Server is there to run and execute the job requests that are submitted to the system. The controller manages them, and they have certain load limits that they can handle.

The Node is the parent class of both the controller and the server, and it contains all the shared methods of the controller and server. It is the parent class because the server and controller are just nodes in the architecture that just have different jobs.

The Server Factory is used for parsing and managing all the servers in the system. It also has the controller information and is a container of servers and controllers. This is the class that maintains the distances between each server or controller in our architecture.

The Request class represents all the requests from the clients. Each request has some basic information, like load amount and location.

The Request Handler Plan is a class that is used for the output of the algorithm. It's a type of data structure, as it contains some key information needed from the load-balancing algorithms. It contains which server the algorithm chose, the controller the algorithm chose, and the time the algorithm took to pick the server and controller. Whenever there is a request, we create a request

handle plan. The chosen server and controller in request handler plan are going to be the ones that will be executing the newly arrived request.

The algorithm interface gets the requests and does calculations to find a server and controller to handle the request. There are two different algorithms that are concrete classes.

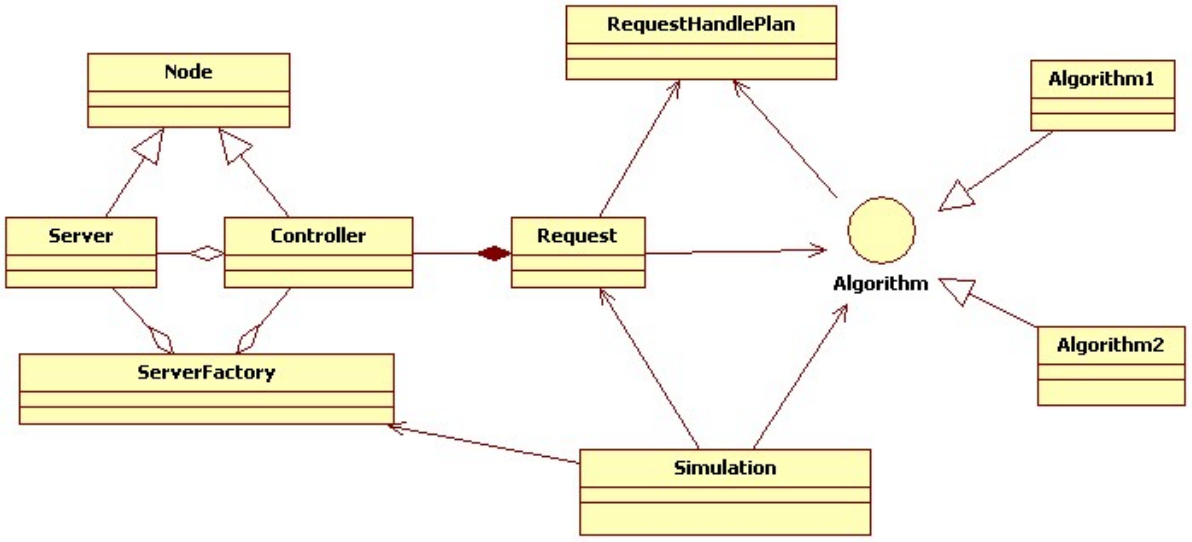
Algorithm 1 is a concrete class to implement algorithm. In this algorithm, we have a somewhat heuristic approach where we weigh each controller based on the amount of load it has with respect to the other controllers, we will tend to choose the one that is least loaded. Then within that controller, we weigh all the servers against one another and then decide which server to pick. This algorithm was found in our research, and we are using this as the benchmark algorithm with which we compare our algorithm's performance to.

Algorithm 2 is a concrete class that implements algorithm. In this algorithm, we look at the amount of load that the new request will take up, and we then look to see which controller's location will be closest to the request's location. These locations are physical locations, and we want to minimize this so that data transfer time in a real distributed system would be minimized. Now, within this closest proximity controller, we look for the closest server and we check the load on that server to see if it can safely handle the new load from the request. If it can, great, we have found the server to execute the request, if not, then we need to find another server within that controller that can handle the request. If none of the servers within the proximity controller can safely handle the request, then we need to move to a backup controller and search through its servers to find a server that can safely handle the request. Each controller has 2 backup controllers and if after checking the 2nd backup we are still unable to find a server that can handle the request's load, we issue a waiting time or delay for that request until one of the servers has enough space to take the request. We will be comparing the results of this algorithm with the results of the benchmark algorithm to see if we can reduce the number of failures or the amount of time the algorithm take, or both.

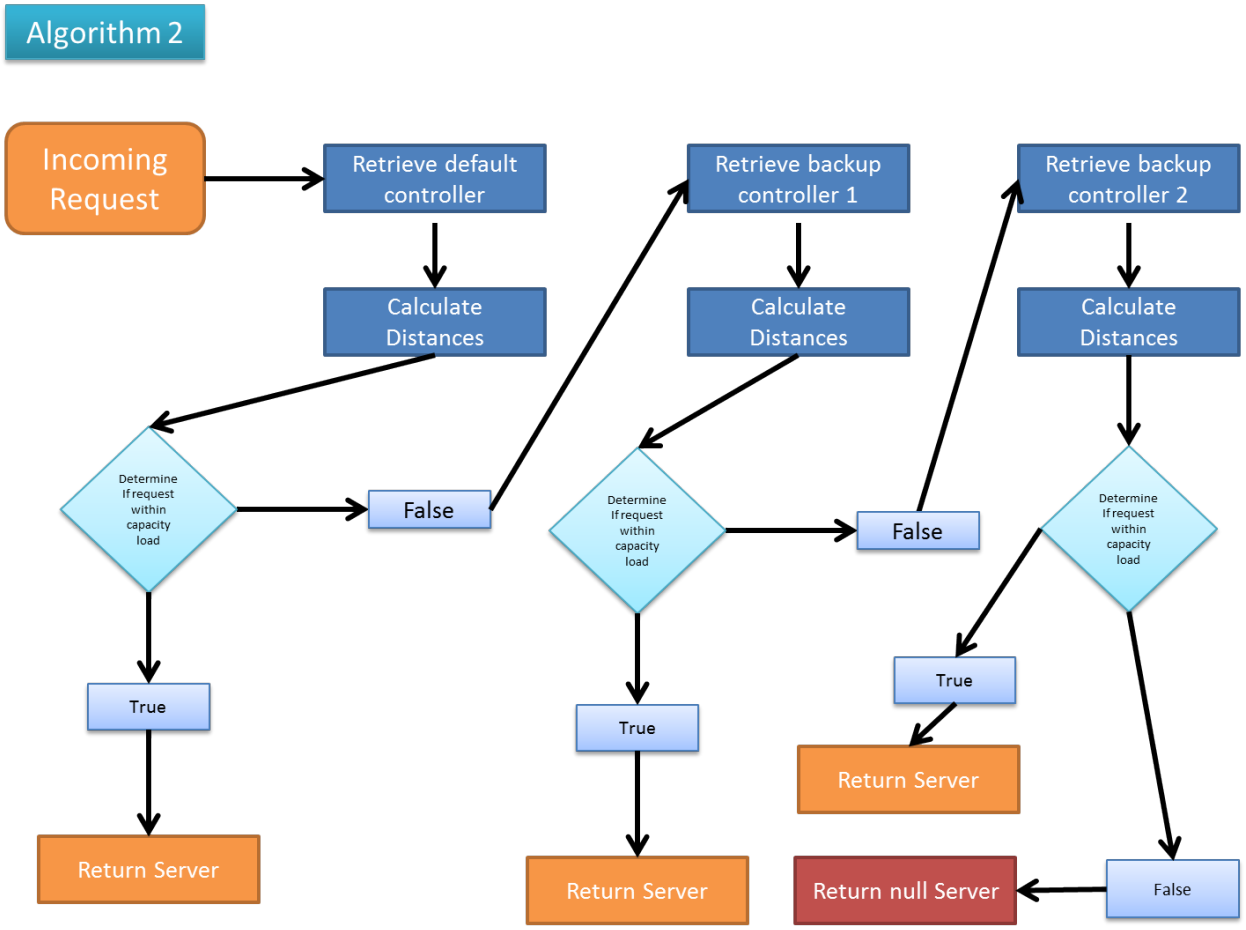
The Simulate class is used to do the simulation for our system. We pass the simulate class our input data, it runs both algorithms, and it provides us with the output data.

Flowcharts

The server factory will generate the nodes from the input data, while the request handler will handle incoming requests. Once a request comes, the assigned controller will determine which server to take the request load. Either algorithm 1 or 2, on separate runs of the data, will do load balancing.



Algorithm 2 calculates the distances between each server and its controller, and assigned server with the shortest distance. If default is full, backup 1 is assigned, and the same calculation happens. If back up 1 is full, then it is sent to back up 2. If all these are full, request will not be assigned a server and it will return a null.



Data Analysis and Discussion

Output generation

The simulation application reads the input data to create the controllers and servers, and then reads the requests to simulate the tasks coming to the controllers and servers.

The controller will run the load-balancing Algorithm 1 or Algorithm 2 to select the server for handling each of the requests. The algorithm will choose the best server and calculate a "decision time" in nanoseconds.

If all the servers, which could handle the request, are busy (full loaded), the simulating application will put the request into a waiting queue and prints out the information. When some server has finished running a task and the freed capacity is enough to handle the first request in the waiting queue, the controller will pull out such request from the waiting queue and assign it to that server.

The simulation application tracks the handling of each request, and prints out: when the request arrives, which server handles it, and the load before and after the server handles the request. The application also sums the "decision time" and "waiting time" for each request and prints out the total "decision time" and "waiting time" in the end of the output.

Output analysis

To analyze the output data we accumulate, we are going to compare two main values. The first value is the algorithm run-time, which is the time needed to complete the load balancing and migration for each request. The second value is going to be the number of errors; these errors are generated from our code when there is a server failure or a failure to complete a job-request.

To analyze that data

To analyze the data, we are going to be comparing the run-time performance of our load-balancing algorithm to a benchmark load-balancing algorithm that was in our research. When the program runs, it first will create sample requests as well as architecture of servers and

controllers. The program will then start executing the requests using the benchmark load-balancing algorithm and the generated requests. After the program is completed using the benchmark load-balancing algorithm, it will then re-run with the same requests but this time, it will use our load-balancing algorithm. Upon completion, we will now have two sets of data, one for each algorithm.

To compare the results, we will compare the run-time for each algorithm. The algorithm with the smaller run-time means that less time was wasted in the program to just decide on which server to pick and migrate the request to that server. Although this is a good way to measure which algorithm is faster, its not a good metric on how effective the algorithm is for load balancing. For this, we will need another metric, which is going to be the number of errors or failures. When we compare the number of errors between the two algorithms we can then see which algorithm was more effective. The effectiveness of an algorithm is measured by having less failures or errors. With these two metrics, we can also see if there is a pattern between the algorithm run-time and the amount of errors. We can see that perhaps spending more time in the algorithm is acceptable if it means that the number of failures will decrease.

Compare output to hypothesis

Our first goal was to create a more stable and "safe" dynamically load-balancing architecture. From our results, we have shown there to be less failures, because we can see a smaller waiting time for jobs that cannot be assigned. This means that our algorithm is very good and no job was unable to find a server for it to run. When compared to the benchmark this is very good because that algorithm had a considerably long waiting time (which means that there were either failures or a proper server could not be found).

Our next goal was to try and make our system deliver a lower average job-response time than the existing algorithms. This goal was not accomplished, unfortunately, because our algorithm was more involved, and had many more lines of code. We are ok with our algorithm taking more time than the benchmark algorithm because the number of failures was much less using our algorithm.

Our last goal was to try to simulate both algorithms using our architecture and compare the response times and results to show that our algorithm was better. We were successfully able to

create a simulation for both algorithms using our architecture. From the comparison of the results we have found that our algorithm, although a bit slower, was more reliable than the benchmark load-balancing algorithm.

Abnormal case explanation

1. When using our algorithm, an abnormal case would be when we get a failure. This can occur for a few reasons, the first being that every server in the entire system was full, but the task was assigned to a server. In our code, we ensured that this could not happen because we would wait before sending the request to the server. We report back to the central directory server (CDS) whether the job will be executed, delayed, or failed. In our algorithm that we created, the requests will not be executed if it will cause a failure.

2. When using the pre-existing algorithm, failures may occur because the algorithm only takes into account the existing loads in the system, and not the load of the new request. It tends to choose the Controller with the lowest overall load. Within that Controller, the algorithm tends to choose the Server with the lowest load. If, for example, a request, which has load much larger than any server could handle, is introduced to this system, it will be assigned a Controller and a Server, and it will attempt to execute on that Server. This would result in an error because the request's load is not considered in this algorithm.

These are the only abnormal cases that could occur in our algorithm, as it is very exhaustive and will not generate any false errors.

Conclusions and recommendations

Summary and Conclusions

To summarize, we have successfully created a new architecture for our load-balancing algorithm. Our architecture contains three key levels, the CDS, Controllers, and Servers. The CDS is responsible for the entire system, and there are 2 of them, which have a heartbeat to keep track of everything. Then there are controllers, which are responsible for a certain amount of servers. The servers are the nodes that will do the computing for all job requests that are sent to the system.

The algorithm that we created first looks at the location of the request and chooses a nearby controller, and then picks the closest server (by location). It checks to make sure that the server can handle the incoming request safely, if it cannot, it moves to a different controller or server until it finds a server that can handle the request. We felt that this is a good way to do the load balancing as it tries to choose the fastest response time (by choosing the closest server).

In Conclusion, we were able to create both our architecture and our load-balancing algorithm and simulate them. We were also able to simulate a benchmark algorithm to compare our results to. In our simulation we found our algorithm to have far less errors or waiting time when compared to the benchmark algorithm. However, this came at the price of having our algorithm take a little bit longer time to run compared to the benchmark algorithm.

Recommendations for future studies

For future studies, we would recommend looking at more types of load balancing algorithms that have more intelligent equations. The ones that we seemed to find were quite simple, either taking an average of the loads, or simply finding the server with the least loads. Not many of these algorithms talked about how they would be keeping track of all the information they were using or what type of architectures their algorithms wouldn't be effective on.

If someone is interested in this topic, and would like to research further into the topic of load-balancing algorithms, we would suggest that they have a strong math background. This is very important when analyzing existing algorithms and their equations or formulas. If one can find

any inefficiency in an existing algorithm, they could task themselves to try and improve that inefficiency.

Another route to go about improving load balancing in distributed systems would be to see if there could be a better architecture for a certain algorithm. Perhaps some parts of the algorithm can become faster if they were implemented on a different type of hierarchy or set-up.

Bibliography

1. Fu Xiao-ling; Xu Ying, "A Load Balance Algorithm for Hybrid P2P Network Model," Computing, Communication, Control, and Management, 2008. CCCM '08. ISECS International Colloquium on , vol.1, no., pp.236,239, 3-4 Aug. 2008

DOI: 10.1109/CCCM.2008.325

2. Li Wang; Shangping Ren; Ke Yue; Kwiaty, K., "Optimal resource allocation for protecting system availability against random cyber attacks," Computer Research and Development (ICCRD), 2011 3rd International Conference on , vol.1, no., pp.477,482, 11-13 March 2011

DOI: 10.1109/ICCRD.2011.5764062

3. Mcheick, H.; Mohammed, Z.R.; Lakiss, A., "Evaluation of Load Balance Algorithms," Software Engineering Research, Management and Applications (SERA), 2011 9th International Conference on , vol., no., pp.104,109, 10-12 Aug. 2011

DOI: 10.1109/SERA.2011.46

4. Mehta, M.A.; Agrawal, S.; Jinwala, D.C., "Novel algorithms for load balancing using hybrid approach in distributed systems," Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on , vol., no., pp.27,32, 6-8 Dec. 2012

DOI: 10.1109/PDGC.2012.6449786

5. Mirrezaei, S.I.; Shahparian, J.; Ghodsi, M., "A topology-aware load balancing algorithm for P2P systems," Digital Information Management, 2009. ICDIM 2009. Fourth International Conference on , vol., no., pp.1,6, 1-4 Nov. 2009

DOI: 10.1109/ICDIM.2009.5356795

6. Radojevic, B.; Zagar, M., "Analysis of issues with load balancing algorithms in hosted (cloud) environments," MIPRO, 2011 Proceedings of the 34th International Convention , vol., no., pp.416,420, 23-27 May 2011

7. Rajavel, R., "De-Centralized Load Balancing for the Computational Grid environment," Communication and Computational Intelligence (INCOCCI), 2010 International Conference on , vol., no., pp.419,424, 27-29 Dec. 2010

Appendices

Program Source Code with documentation

LoadBalance.zip: Source code

That file contains all of our source code with documentation.

Input/output listing

InputData.xlsm: Input data generator

InputData.txt: Input data

aaa1.txt: Output for algorithm 1

aaa2.txt: Output for algorithm 2

These files are the names of the input and output files.