Project Report

on

Survey on Load Balancing in Distributed Systems

CS550 – Advanced Operating Systems

Submitted By :

Amit Nikam (A20470263)

Guided By :

Professor Xian-He Sun

INDEX

Abstract

The behavior of user in a distributed system is difficult to characterize and so the management of resources and applications is a very complex task. When the demand for computing power in the system increases, it becomes important to balance the load in the system properly. Load Balancing is the process of distributing the load among various nodes of a distributed system to improve both job response time and resource utilization while also avoiding a situation where some of the nodes are heavily loaded while other nodes are idle or lightly loaded.

In this survey I evaluate and characterize the implementation methods and algorithms involved in load balancing, and also propose a future scope for this topic.

1 Introduction

1.1 Project Name

Survey on Load Balancing in Distributed System.

1.2 Goal

The goal of this survey is to do a detailed evaluation and comparitive study on different aspects of load balancing. This study shall be serve as a practical guideline to researchers and developers in designing efficient load balancing algorithms and implementations for distributed computing systems.

1.3 Problem Statement

Distributed computing system provides high performance environment that are able to provide huge processing power using multiple nodes. [1] Thus chances are that some nodes will be idle while the other are overloaded. By implementing load balancing it is possible to make every processor equally busy and to finish the works approximately at the same time. Load balancing is the way of distributing load units across a set of processors which are connected to a network which may be distributed across the globe. Some benefits of doing this are reduced downtime, scalability, flexibility and efficiency. [2, 3, 4]

As a distributed systems researcher or developer, it may become difficult to decide which load balancing policies and algorithms to implement since optimal implementation is completely dependant on the architecture and nature of the distributed system. [5] It is challenging to achieve an optimal balance. If not implemented properly, improving the system wide load balance comes at additional cost, overheads and performance issues. Thus it becomes very important to have lucid understanding of the concept of load balancing.

Furthermore, load balancing can be implemented at Application level, OS level or the Network level. [6, 7, 8, 9, 10, 11] Thus, the type of load balancing to implement can be difficult to decide upon and so it becomes important to study these implementation methods along with algorithms.

Thus, we would be surveying the load balancing implementation types and algorithms (methods).

2 Implementation Types

A significant amount of work has been done on load balancing services at various system levels. This includes research on load balancing services at the network, the operating system, and the middleware level. But before we look into these services at different levels, we need to understand the form in which a load balancer might exist.

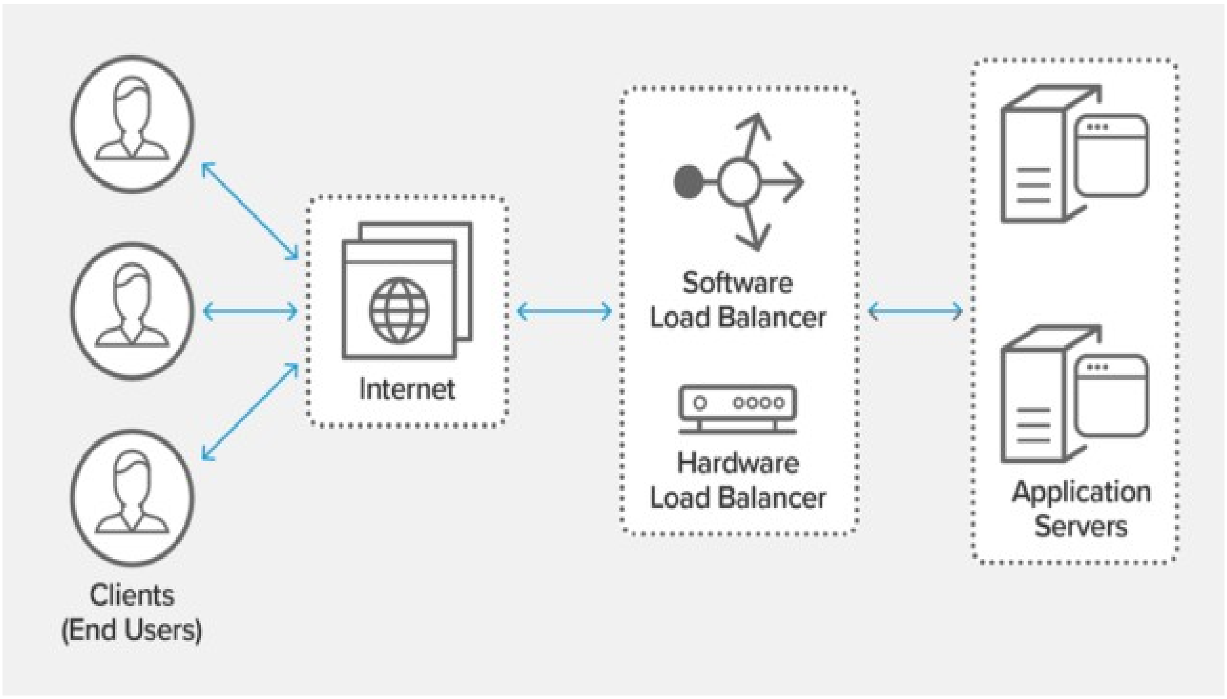

Figure 1: basic implementation architecture of a load balancer [12]

A load balancer is incorporated in a layer which is between the user access (mostly internet) and the application infrastructure known as the application delivery controller (ADC). Within this layer, the load balancer can either be a software or a hardware; or today can even be an affordable cloud based solution, for example, AWS Elastic Load Balancing. [13] So lets understand these forms:

- **Hardware Load Balancer Device (HLD)**

  HLD are a physical appliance often referred as specialized routers or switches which are deployed in between the servers and the client. It can also be a dedicated system(ADC) in between the the client and the server to balance the load. The hardware load balancers are implemented on Layer4 (Transport layer) and Layer7 (Application layer) of OSI model. Some prominent among these hardwares are L4-L7 routers. These can often be expensive.

- **Software Load Balancer**

  Software load balancers generally implement a combination of one or more scheduling algorithms and come in two forms - commercial or open-source - and must be installed prior to use. Like cloud-based balancers, these tend to be more affordable than hardware solutions.

Now that we have an understanding of the forms of load balancers, lets do a comparitive study on their implementation levels and characterize them. In next three sub-sections we will look at the different types implementations and in the fourth sub-section we will compare them based on characteristics.

2.1 Network-based load balancing:

Network-based load balancing services are for most parts implemented using HLDs, and as such, decisions are based on the frequency at which a given site receives requests [6]. For example, routers [7] and DNS servers often perform network-based load balancing.

Load balancing performed at the network level has the disadvantage that load balancing decisions are based solely on the destination of the request. The content of the request is often ignored. This form of load balancing also makes it difficult to select the load metric to be used when making balancing decisions and thus they are inflexible.

But due to the use of HLDs, this implementation is highly efficient and can really be beneficial if implemented in specific cases like load balancing the traffic at a much high geographical level (or even global level).

2.2 OS-based load balancing:

Load balancing at the operating system level [8, 9, 10] has the advantage of performing the balancing at multiple levels of OSI model. These can either be in hardware form(HLD) or software form. This type of balancing is essentially transparent to a distributed application.

However, it suffers from many of the same problems that network-based load balancing suffers from, such as inflexible load metric selection and not being able to take advantage of request content. OS-based load balancing may also be too coarse-grained for some distributed applications where it is the objects residing within a server, rather than the server process itself, that must be load balanced.

Although the advantage here is that, even though it is less efficient than network-based load balancing, it is much more efficient than middleware-level load balancing. Infact due to the slightly higher flexibity than network-based load sharing, this proves to be useful in many cases.

2.3 Middleware-based load balancing:

Middleware-based load balancing provides the most flexibility in terms of influencing how a load balancing service makes decisions, and in terms of applicability to different types of distributed applications [11]. Load balancing at this level provides for straightforward selection of load metrics, in addition to the ability to make load balancing decisions based on the content of a request.

The disadvantage is that, this is mostly software based and implemented above other OSI layers and thus involves alot more overhead and delay i.e. it is slow / less efficient compared to a Network-based or OS-based load balancer.

2.4 Characterization of Implementation Types

Now that we have understood the implementation types and forms of load balancers, let's characterize them in the following table.

Type	Network-based	OS-based	Middleware-based
Decision Basis	The frequency of received requests	Depends on implementation level	Multiple possible implementations
Load Metric Selection	Inflexible	More flexible than Network-based, but less than middleware-based	Highly Flexible
Efficiency	Highest	Higher	High
Form of load balancer	Hardware Based	Hardware + Software (depending on level implemented at)	Software based (but can have hardware integrated to middleware)
Advantage	High Efficiency	Can perform balancing at multiple OSI level	Ability to make decision based on the content of a request
Disadvantage	Dicisions are based solely on destination of the request	Not being able to take advantage of request content	Significant overhead involved, thus less efficient than other two

Now that we are done characterizing implementation, let's look at a few algorithms.

3 Load Balancing Algorithms

There are multiple load balancing algorithms that have been developed due to their unique advantages over others, although there is no single algorithm that is suitable for all needs.

Load balancing algorithm can be implemented to make decisions based on nature of application, balancing quality, load generation patterns, hardware parameters, location, distritbution or other such criterias. Thus an algorithm can be preemptive or non-preemptive; and can be stateful (requiring information on nodes) or stateless.

Regardless of what all features it has, the goal is to achieve optimal load balancing. That is, the best possible performance. Although this is not empirically possible in all cases all the time, and so in this case we find a stable sub-optimal solution.

In this section’s sub-sections we will be looking at the categories of algorithms, study a few algorithms, compare the studied algorithms and then finally will characterize the categories based on the comparitive study done on these algorithms.

In general the algorithms can be classified into two categories [5] : static and dynamic. Let’s understand what classifies each algorithm into one of these categories.

3.1 Categories of Algorithms

3.1.1 Static Load Balancing

In static algorithm the processes are assigned to the processors at the compile time according to the performance of the nodes. Although the current performance of the nodes is not known by the balancer as static algorithms do not collect any information about the nodes.

Once the processes are assigned, no change or reassignment is possible at the run time. Number of jobs in each node is fixed in static load balancing algorithm. The assignment of jobs is done to the processing nodes on the basis of the following factors: incoming time, extent of resource needed, mean execution time and inter-process communications.

Since priori knowledge of these parameters is required, they are often called probabilistic algorithm. But if accurate information and resources related to system are known, optimal solution can be achieved, given, nodes are stable. If all the information and resources related to a system are known optimal static load balancing can be done.

As there is no migration of job at the runtime like in dynamic algorithms, a comparitively little overhead occurs.

3.1.2 Dynamic Load Balancing

During the static load balancing too much priori information is required, which is not always available. This is where dynamic load balancing algorithm become useful.

The distributed system is for most parts heterogeneous, i.e. each node might differ from other in terms of speed, memory sizes, communication, etc and thus dynamic algorithms tend to perform better in distributed system. In this the assignment of jobs is done at the runtime, depending upon the situation that is the load will be transferred from heavily loaded nodes to the lightly loaded nodes.

Since the information needs to be collected and load needs to be transferred from one node to another, significantly more overhead occurs. This strategy collects the information about the system state and about the job information. As more information is collected by an algorithm in a short time, potentially the algorithm can make better decision [14].

3.2 Study of Algorithms

Now that we have understood the two categories of the algorithms, let’s look at some of the algorithms which are widely used. In this sub-section we will understand the workings of these algorithm and in the next sub-section we will compare these algorithms based on type, state information and performance.

To begin with, lets look at the top five most common load balancing algorithms.

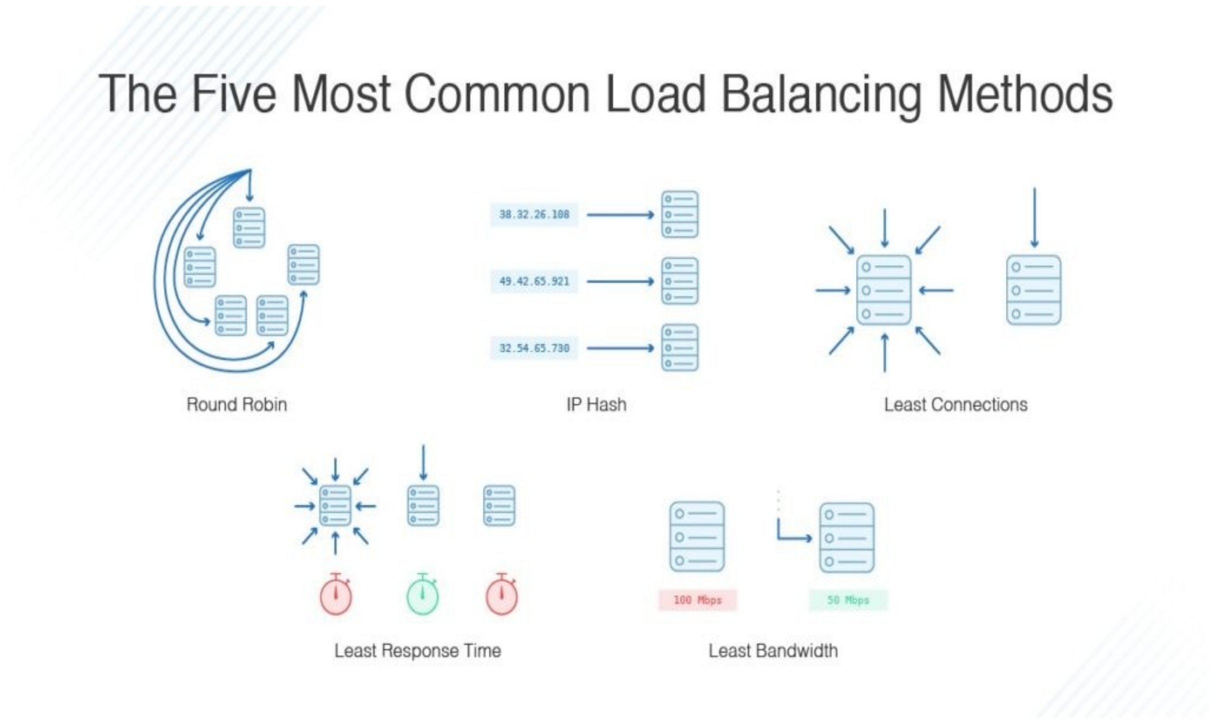

Figure 2: top five most common load balancing algorithms [15]

1. **IP Hash:** [16]

   In this method, the load assignment decision is based on the pre-defined IP to offload server mapping in the Hash table available at the load balancer. Thus, whenever a request is received, depending on the IP, it is forwarded to the respective server.

   This algorithm is a static algorithm as the balancer is not aware of the states of the server nodes and the decision is based purely on pre-defined rule. This algorithm tends to have an excellent performance as while implementing the developers are often well aware of which IP locations most requests come from, and so, they map the servers in such a way that the load is evened out.

2. **Round Robin:**

   This algorithm is one of the simplest to implement and scale. Infact most applications can get optimal performance by just implementing Round Robin algorithm.

   In this algorithm, as visible in Figure 2, every next request is assigned to the next server until all the servers are assigned load. If such is the case, then the assignment starts from the beginning again, and keeps repeating in a cyclic form, thus the name Round Robin.

   Although this gives excellent performance, it is to be noted that if a server among the collection is slower than the cycle speed of the load balancer, it may happen that that particular slow server might start missing loads.

   This happens to be a static load balancing algorithm as no state information is requested by the load balance to make the decision.

3. **Random:** [2]

   This, just like Round Robin and IP Hash is a static algorithm. Just as the name suggests, the load balancer randomly assigns the load to one of the servers. This algorithms performs well, but not exactly as good as the previous two.

   This is due to the fact that at low number of server, chances are only one server might get randomly chosen and overloaded. On the other hand in a scenario where there are too many servers, it can happen that some of the servers are getting ignored completedly.

   This sort of unstable behaviour is not desirable for a load balacing implementation.

4. **Least Connections:** [16]

   In this algorithm the load balancer needs to know the number of connections that a particular server has. Based on this information, the load balancer makes a decision to assign jobs or not. Although it is not completely aware of the state of the server, it has somewhat information, and thus it is partially stateful.

   That is, this algorithm is a dynamic algorithm. This algorithm tends to perform better than random assignment algorithm, because at least no servers are left idle. Even so, it is not considered to have excellent performce in distributed systems because the servers in distributed systems are heterogeneous, i.e. they have different performance, memory, communication speeds and so on.

   Thus when the balancer assigns the load to a server with least connections, chances are the load assigned there might take longer to process than a server with more connections because of difference in performance.

5. **Least Response Time(SED):** [4]

   In this algorithm, the jobs behave in a manner that is considered selfish i.e. in their best interest. The job is assigned to the server that has the shortest expected delay i.e. the best performance as it can complete the job in shortest time(delay).

   This algorithm works really well at less loads, but as the load increases, chances are the fastest server might get bombed with requests and multiple of them will end up in it’s waiting queue. If a request is not accepted, the job is sent to the second fastest expected server.

   None the less, to be able to know the server with least delay, the balancer has to communicate with the nodes about their states. This involved alot of overhead cost and also makes this algorithm dynamic. Although this is a good way of getting the job done(with least response time), it is not the best.

6. **Modified SED:** [4]

   In this varient of SED (extension of the previous algorithm), we don’t just consider processing time in delay, but also consider the communication delay. Communication delay is defined as the time a job takes to reach its destination and end of queue i.e. it’s time to execution is considered along with its execution time.

   To be able to implement this, alot of information on the state of the server is needed and so this is a dynamic algorithm just like SED. This performs better than SED because now other servers are also considered during the decision making instead of just the fastest performing one. This distributes the load evenly and also reduces the response time as the waiting time, which was originally spent in communcation and queue is now reduced.

7. **Least Bandwidth:**

   In this algorithm the server which is the closest, i.e. has least bandwidth is allocated the load. If the server receiving the load is overloaded, the load is forwarded to the next server that is nearest to it.

   In this implementation complete state information is not required by the load balancer, just the information about server with least bandwidth is enough. This dynamic algorithm performs better than some of the algorithms because the overhead involved is less and also the response time are often good.

8. **No Queue (NQ):** [4]

   In this algorithm, the job is assigned to the server that has not queue, i.e. the one that can immidiately start the task. Incase no such server is available, the job is assigned to the server that can complete it in the shortest time. Incase that server is overloaded, then the next best server gets the job.

   This is a dynamic algorithm that does good. There is a lot of overhead involved in this algorithm which reduces its performance.

9. **Modified NQ:** [4]

   In this modified version, the balancer only looks from a set of nodes to find the node with no queue. This set is a set of nodes that can be extected to have no queue. Thus doing so improved the performance.

   Nonetheless, in case there is no such server with no queue, the server with least delay(communcation delay considered this time) is assigned the job. This way performance is promised.

10. **Greedy Throughput:** [4]

    In this algorithm, the objective is to maximize the throughput of the system i.e. to compute most tasks possible between the consecutive tasks. In this, all the servers are well utilized and this it is considered to be efficient. The overall response time gets reduced in this dynamic algorithm.

11. **Central:**

    Central load balancer algorithm happens to be one of the most used algorithms. The manager and worker form of this algorithm fits well in the distributed system’s architecture. In this, the central node received communication about the state of the worker node by the respective worker node. Then this load balancer(manager), from time to time keeps updating the network with all the information it has.

This has an excellent performance, but sadly has a central point of failure i.e. if the load balancing node fails, the entire system fails. Still, this dynamic algorithm can prove useful for many usecases, specially the ones where data parallelism needs to be achieved over the distributed network.

12. **Broadcast:**

    In this, each of the nodes broadcasts their state to all other nodes. A huge overhead is involved and performance of this algorithm is poor. Thus it is not often used. This proves that not all dynamic algorithms are good when it comes to performance.

3.3 Compare & Analyze Algorithms

Now that we have understood some of the most used algorithms for load balancing, lets do a comparison between them to better understand their performance as compared to each other.

Algorithm	OverHead Involved	Type	State Information	Performance
IP HASH	Neg	Static	No	Excellent
Round Robin	Neg	Static	No	Excellent
Random	Neg	Static	No	Good
Least Connection	Low	Dynamic	Partial	Better
Least Response Time(SED)	High	Dynamic	Yes	Good
Modified SED	Moderate	Dynamic	Yes	Better
Least Bandwidth	Low	Dynamic	Partial	Better
No Queue (NQ)	High	Dynamic	Yes	Good
Modified NQ	Moderate	Dynamic	Yes	Better
Greedy Throughput	Moderate	Dynamic	Yes	Better
Central	Low	Dynamic	Yes	Excellent
Broadcast	Extreme	Dynamic	Yes	Poor

Through this comparitive study it is observed that dynamic algorithms are capable of performing just as good as static algorithms. Infact for distributed systems where the system is heterogeneous, dynamic algorithms tend to perform better.

Furthermore, it is observed that as the overhead/communcation increases in dynamic algorithms, the performance decreses. Thus, when implementing dynamic algorithms, the aim should to reduce overhead as much as possible to get the best possible performance so as to acheive the sub-optimal implementation.

3.4 Category Characterization

Now that we have completed our comparitive study on the algorithms, let's characterize the two categories of the algorithms using our findings. This characterization will serve as a guidance when deciding which category of algorithm to implement.

Parameters	Static Algorithms	Dynamic Algorithms
Reliability	Less	More
Complexity	Less	More
Performance	Less	More
Flexibility	More	Less
Stability	More	Less
Implementation	Easy	Difficult
Overhead Involved	Negligible	More
Resource Utilization	Less	More
Cost	Less	More
State woggling	No	Yes
Scalability	Yes	Not Always

It can easily be observed that Static Algorithms have characteristics which makes it easy and attractive to implement at the same time. In applications where operations are fixed and well known, static algorithms can even achieve optimal solutions. Infact even without priori known, implementing the static algorithms suffices the needs of most of the systems, which is why they are widely accepted, specially in many web servers. Nonetheless as the traffic grows and system performance requirements go higher, it becomes important to update the load balancing methods.

That concludes our study part for the survey, in next section we would be understanding what according to me is the possible future scope for load balancing in distributed systems as a topic.

4 Future Scope – Application Specific Load Balancing

As the applications are becoming more and more data-driven, the need to perform computationally intensive tasks has increased significantly. Thus the need for application specific load balancers is high today.

The goal of such load balancer would be to minimize the time required for computationally intensive tasks rather than the average time of all task for a specific use case. These computational tasks could involve tasks which would otherwise require a super-computer; for example: Distributed Image Processing, Financial Modeling, Quantum Chemistry, Astronomy, Weather Modeling, Prime Number Factoring and so on.

Distributed Systems are mainly heterogeneous in nature, and thus application specific load balancing algorithms need to be dynamic in nature to adjust to the environment so as to achieve their goal. Since the nodes in this system would be processing the data parallely, it would make it a Data Parallel Distributed System.

One of the challenges of implementing such an application specific load balancer is that it needs to be scalable and highly performing. Furthermore, it not only requires us to have good knowledge of the system, but also the domain of the application.

To address these challenges, we would be discussing a really promising load balancing implementation which was published as a patent by IBM in 2010 for their super-computer Blue Gene /L. [17] Although we will use this just as an example, but a similar sorts of load balancer to this patent with scalability and performance in mind can be implemented and could also work well for distributed systems.

Infact, our example of data parallel image processing implementation [3] can be integrated with a load balancing implementation of this sorts to build an applicaiton specific load balancing that’s scalable and highly performing.

4.1 Example: Data Parallel Image Processing [3]

In this example, an image is convolved with a filter of kernel size K. As the process of convolution in Computer Vision happens to be a serial task, i.e. the image is convolved from left to right and then from top to bottom, each compute worker will have a maximum of two neighbors at any given point.

Here compute workers do the actual processing of convolution(s) while one node among them acts as a manager and overlooks the work between a pool of workers. Thus, the worker nodes perform the compute intensive task and are also capable of communicating among each other.

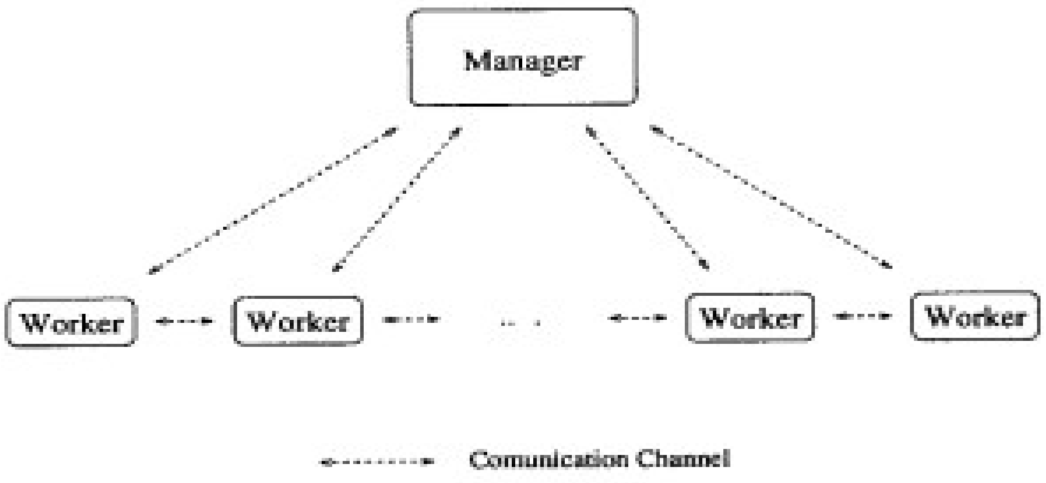

Figure 3: layout of workers and manager

The image to be processed is broken down into sub-images and these patches are provided to the respective workers that will be performing convolution on them. As this is implemented over heterogeneous system, naturally some workers will be slower than others.

To achieve maximum computational throughput, the goal here would be to process image patches in such a manner that we minimize the time required computationally intensive tasks rather than the average time of all task for a specific use case. Therefore, the system will perform computation in such a way that the computations that have the least chance of being transferred will be processed first.

Therefore for left most worker, the processing will start from the left frame. Meanwhile for right most worker, it would start from the right. While for rest of the workers the computation would start at the center. Furthermore, the dynamic load balancer will time to time check processing status with the workers.

As soon as manager realizes that one of the processing nodes is slow, it will change the target processing window for its adjacent node in such a way that the adjacent fast nodes will communicate and fetch a few frames to process, thus minimizing the processing time. The figure 4 rightly represents the operation.

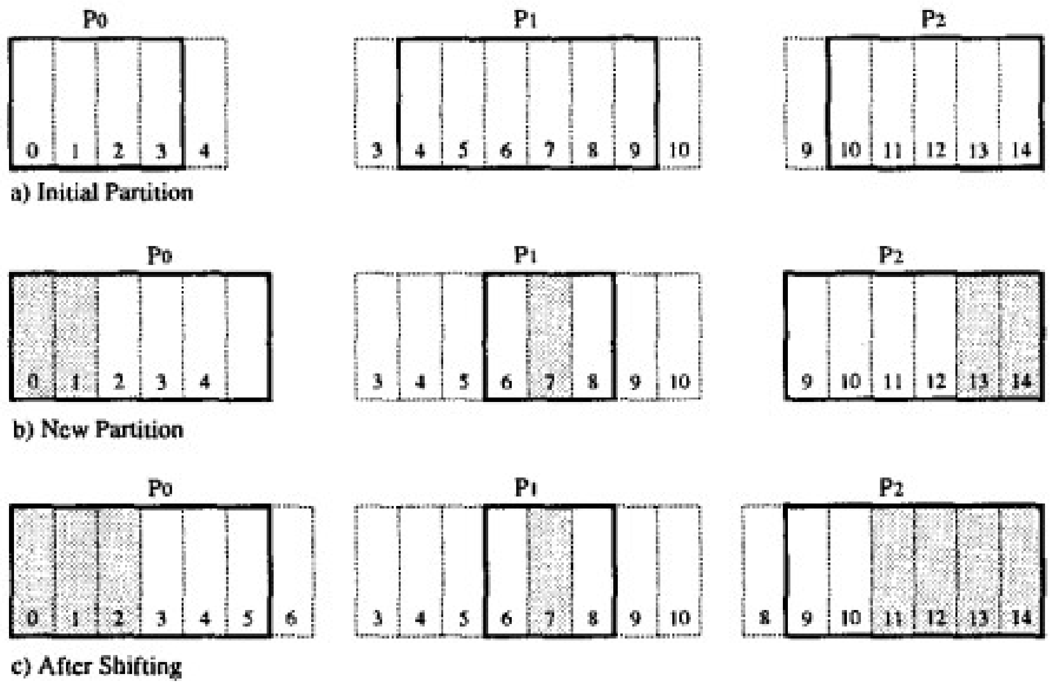

Figure 4: breakdown of image chunks and processing patterns

Thus by implementing the load balancing implementation similar to that seem in IBM's patent paper where a Coordinator overlooks multiple managers, this scale of operation can be increased so as to either processor more images at one time or to process the same image much faster or to process higher resolution images much faster. The following figure represents the IBM's implementation.

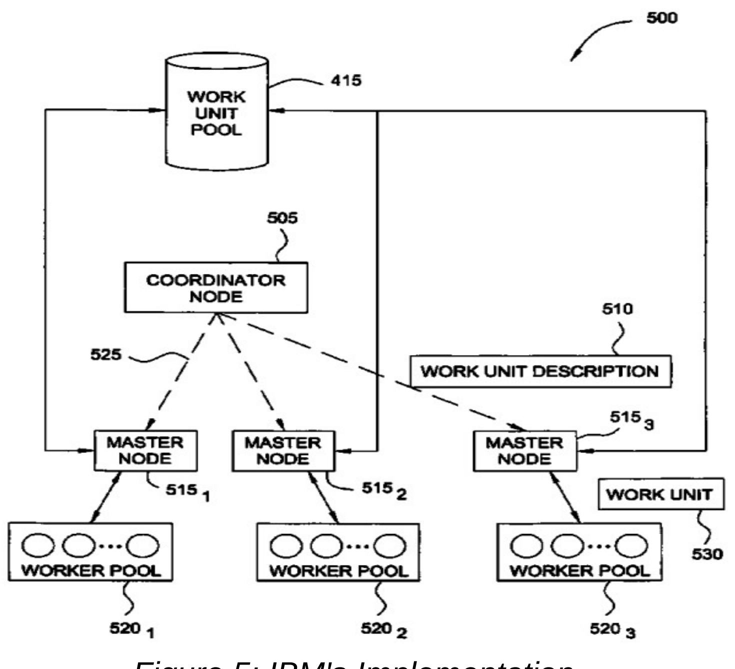

Figure 5: IBM's Implementation

5 Conclusion

Since load balancing in distributed systems is a topic just as old as distributed systems itself, it evolves as the distributed systems evolve. This happens to be a active area in research today as the demand for heterogeneous computing due to the wide use of internet has increased.

As the applications are becoming more data-driven, the need for high performing systems is felt. As the performance of the distributed system is directly dependent on how well the load balancing is implemented, it becomes important to understand this topic inside-out. This is exactly what we did in this survey project.

In this survey, we did a comparitive study on the different implementation types for a load balancer and also did a comparitive analysis on the different algorithms and their categories. Throughout the study we grouped and characterized our learnings.

We did not stop there, we propose a future scope for this research topic and also backed it with an example wherein we explored the possible implementation of a load balancer for performing compute intensive tasks, and discussed how it is different from the general purpose load balancers.

This survey not only provides an insight view of the load balancing as a topic, but also offers practical guidelines to researchers and developers in designing efficient load balancing algorithms for distributed computing systems.

6 References

[1] Lüling R., Monien B. and Ramme F. “Load Balancing in Large Networks: A Comparative Study. In Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing”. CiteSeer x , 1991.

[2] Berenbrink P., Friedetzky T. and Steger A. “Randomized and Adversarial Load Balancing”. CiteSeer x , 1997.

[3] Hamdi M. and Lin C.K. “Dynamic Load Balancing of Data Parallel Applications on a Distributed Network”. In 9 th International Conference on Supercomputing, ACM, 170-179, 1995.

[4] Kabalan K.Y., Smari W.W. and Hakimian J.Y. “Adaptive load Sharing in Heterogeneous system: Policies, Modifications and Simulation”. CiteSeer x , 2008.

[5] Md. Firoj Ali and Rafiqul Zaman Khan. “The Study On Load Balancing Strategies In Distributed Computing System”. International Journal of Computer Science & Engineering Survey (IJCSES) Vol.3, No.2, April 2012.

[6] E. Johnson and ArrowPoint Communications, “A Comparative Analysis of Web Switching Architectures.” http://www.arrowpoint.com/solutions/white papers/ws archv6.html, 1998.

[7] Cisco Systems, Inc., “High availability web services.” http://www.cisco.com/warp/public/cc/so/neso/ibso/ibm/s390/mnibm wp.htm, 2000.

[8] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Leonard, and W. Neuhauser, “Overview of the CHORUS Distributed Operating Systems,” Tech. Rep. CS-TR-90-25, Chorus Systems, 1990.

[9] D. Ridge, D. Becker, P. Merkey, and T. Sterling, “Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs,” in Proceedings, IEEE Aerospace, IEEE, 1997.

[10] W. G. Krebs, “Queue Load Balancing / Distributed Batch Processecing and Local RSH Replacement System.” http://www.gnuqueue.org/home.html, 1998.

[11] T. Ewald, “Use Application Center or COM and MTS for Load Balancing Your Component Servers.” http://www.microsoft.com/msj/0100/loadbal/loadbal.asp, 2000.

[12] NGINX, https://www.nginx.com/wp-content/uploads/2014/07/what-is-load-balancing-diagram-NGINX-640x324.png

[13] Amazon Web Services, https://aws.amazon.com/elasticloadbalancing/?elb-whats-new.sort-by=item.additionalFields.postDateTime&elb-whats-new.sort-order=desc

[14] Jain P. and Gupta D. “An Algorithm for Dynamic Load Balancing in Distributed Systems with Multiple Supporting Nodes by Exploiting the Interrupt Service”. Academy Publisher, 232-236, 2009.

[15] DNSStuff, https://www.dnsstuff.com/wp-content/uploads/2020/01/the-five-most-common-balancing-methods-1024x536.jpg

[16] I. P. A. Suwandika, M. A. Nugroho and M. Abdurahman, "Increasing SDN Network Performance Using Load Balancing Scheme on Web Server," 2018 6th International Conference on Information and Communication Technology (ICoICT), Bandung, 2018, pp. 459-463, doi: 10.1109/IcoICT.2018.8528803.

[17] Archer C.J., Mullins T.J, Sidelnik A. and Smith B.E. “Parallel Computing System Using Coordinator and Master Nodes for Load Balancing and Distributing Work”. United State Patent, 2010.