The State of Load Balancing: A Literature Review

Gabriel Dos Santos
Yale University

Abstract

This literature review investigates load-balancing techniques in distributed systems. It compares static and dynamic algorithms as well as cooperative and non-cooperative approaches. Additionally, the paper explores the implementation of load-balancing techniques in popular technologies such as Maglev and Hadoop. The study analyzes the advantages and limitations of each method, considering factors such as system efficiency, fault tolerance, and scalability. Overall, the paper provides a comprehensive review of load-balancing techniques in distributed systems, aiming to aid developers and researchers in selecting the most appropriate approach for their specific needs.

1 Introduction

Load balancing is the process of distributing incoming workloads across multiple back-end resources. As we enter the globalized age of cloud computing, load balancing becomes more important than ever. In the setting of distributed systems, load-balancing algorithms help decide how to distribute the workload among available resources. They may use information on node capacity, network conditions, and workload type to help determine the optimal strategy. Overall, effective load balancing will improve system performance, increase system availability, and reduce response times.

Load balancing is responsible for resource provisioning and task scheduling in distributed systems. It has the goals of providing high availability of resources, efficient utilization on varying workloads, energy reduction, and lower costs [9]. Various methods with different trade-offs achieve these goals.

In this literature review, we will explore the various types of load-balancing algorithms. We will dissect load balancing by category and dive deep into numerous interesting developments. Later, we will analyze specific technologies in cloud-based load balancing. In the end, we will have an encompassing view of the state of load balancing in distributed systems.

2 Load Balancing Algorithms

First, we will explore the differences between static and dynamic algorithms. Then we will analyze various approaches such as global, cooperative, and non-cooperative.

2.1 Static vs. Dynamic

2.1.1 Static

Static algorithms are the simplest type of load-balancing algorithm. They often divide workload equally to nodes based on allocated weights [3]. All the decisions are fixed and made during compile time. Hence, they are independent of changing system conditions.

The Round Robin load balancer is a classic example of a static algorithm. It works by assigning tasks to each node in a circular manner. A main advantage is that this algorithm does not require inter-system communication. However, it is not expected to achieve good performance in a general case [10].

Static techniques struggle in the presence of varying workloads. For example, one node may be given a relatively long task to complete. The static scheduler does not take this into account and queues another task to the node. Therefore, some nodes may be stuck processing heavy loads meanwhile others are processing no requests. Hence, we will not be focusing on static algorithms as they have many inefficiencies.

2.1.2 Dynamic

Dynamic load balancing differs from static load balancing in its ability to adapt to system conditions. They utilize system information such as CPU queue length, CPU utilization, or idle time to guide load balancing.

There are two main approaches to dynamic load balancing: distributed and centralized. In a distributed approach, each node exchanges information with every other node in the system to make load-balancing decisions. This approach has the advantage of fault tolerance, meaning that failure of a single node in the system will not cause the entire operation to

halt [1]. However, it can generate a large number of messages and create a burden on the communication system.

On the other hand, in a centralized approach, there is a central entity that collects load information from all nodes and makes load-balancing decisions. This approach has the advantage of fewer messages being exchanged, but it may have lower fault tolerance and can create a bottleneck in the system [1, 8].

The dynamic load balancing process also has several components. First, new requests go through the transfer policy where the node decides whether it will keep the task or redirect the job. This uses information about the system state. The information policy provides the relevant data on system nodes to guide load balancing decisions [1, 8].

The location policy follows when the transfer policy decides the move the job. Different load-balancing techniques can be used such as random, probing, and negotiation. In the random technique, a node will randomly move jobs to another node if its load (CPU queue length) is too high. In the probing technique, a random subset of nodes is selected, and the best node is chosen based on the information gathered. In the negotiation technique, nodes communicate about their status and decide to receive more load from an overwhelmed node [1].

Static algorithms use fixed rules that require prior information on the system state; meanwhile, dynamic algorithms are flexible and use live information on the state of the system. Overall, dynamic algorithms have the potential to be better but come at a higher implementation complexity cost.

2.2 Approaches

There are various approaches to dynamic load balancing algorithms which include, global, cooperative, and non-cooperative. The latter types of approaches are based on game theory which is the study of strategic interactions between rational players.

2.2.1 Global Approach

In the global approach, there is one central decision maker for the entire system [5]. This resembles the centralized dynamic approach mentioned previously. The decision maker optimizes for the entire system and thus is simple but lacks fault tolerance.

We will focus on the more interesting decentralized algorithms that use cooperative and non-cooperative game theory.

2.2.2 Cooperative Approach

In contrast, the cooperative approach has decision-makers that cooperate to reach an optimum. They must communicate and make joint agreements on load balancing. The goal of their cooperation is to reach a Pareto efficient point such that no other strategy improves latency for all load balancers [5, 6].

One example of a cooperative load-balancing algorithm is COOP. They treat the load-balancing problem as a game with players having the goal of minimizing their execution time. To calculate the load distribution, they first sort nodes in decreasing order of processing rates. They then create a weight from each computer's processing rate and the overall system processing rate. The problem is typically NP-hard, however, they obtain a $O(n * log(n))$ algorithm by assuming the game has a convex objective function [5]. COOP outputs the load allocations for each node and ensures that all jobs receive fair treatment independent of the allocated node.

COOP performs much better than static proportional schemes like a round robin at medium loads. A unique property built into all of these game theory equilibriums is that they ensure 100% fairness for all tasks independent of the machine. Many peer algorithms like the shortest expected delay scheme (DYNAMIC) decline in fairness as the load increases. [5].

2.2.3 Non-Cooperative Approach

Similar to cooperative algorithms, the non-cooperative approach also has several decision-makers but they do not communicate. Each job tries to optimize its own response time independent of other nodes. By each load balancer trying to optimize itself, it results in an equilibrium where the entire system is being optimized.

Some examples are the OPTIMAL [2] and NASH [6] algorithms which generate weights similar to COOP by using each node's processing rate and job arrival rate. They make assumptions in regards to positivity, conservation, stability, convex objective functions, and Poisson task arrivals [2, 6]. Some unique features to note include that the algorithm is periodically refreshed when the system state changes to maintain efficiency.

NASH performs similarly to peer algorithms like global optimal schemes (GOS) at all utilization levels. NASH maintains full fairness meanwhile other schemes like GOS drop in fairness index as system utilization percentage increase [6].

3 Load Balancing Technologies

3.1 Maglev

Maglev is a fast and reliable software network load balancer. Google created this load balancer because they receive millions of queries per second and must meet this high demand at low latency. Maglev provides reliable packet delivery despite unexpected failures and frequent changes.

3.1.1 System Design

Maglev is responsible for announcing VIPs to the router and forwarding VIP traffic to the service endpoints. It uses direct server return to send responses directly to the router so that Maglev does not need to handle returning packets which are typically larger [4].

Forwarding packets requires efficiency and consistency. First, the packets are run through a 5-tuple hash into the connection tracking table. If no entry is found for the given hash, a new back-end is selected for the packet [4]. Otherwise, the existing mapping to the back-end is re-used to maintain consistency.

3.1.2 Maglev Hashing

Maglev can maintain high connection consistency by combining connection tracking tables with Maglev hashing. Each node has a list of preferences based on unique constant skips and offsets. All nodes take turns filling their most-preferred table positions that are empty. This process has a worst-case $O(M * M)$ time where M is the size of the lookup table [4]. Ultimately, Maglev hashing provides almost equal load balancing and minimal disruption to packets when the back end changes.

3.2 Hadoop

Hadoop is an open-source framework for storage and data processing consisting of the Hadoop Distributed File System and MapReduce respectively. These processes follow the Leader/Follower architecture where the leader is responsible for tracking jobs and the follower is responsible for processing tasks within the job [8]. Map and Reduce divide their jobs for load-balancing purposes.

By default, Hadoop uses static FIFO scheduling for jobs. A job is divided into tasks for each node but only one job is executed at a time on a cluster [8]. This scheme can suffer from poor parallelism and higher latency in the presence of heterogeneous clusters due to load imbalance [7]. Dynamic load balancing algorithms can improve efficiency and fairness by using the information on system state and previous runs to create better execution plans for job-tracking leaders.

4 Conclusion

In conclusion, this literature review provides a high-level analysis of load-balancing techniques in distributed systems, comparing static and dynamic algorithms as well as cooperative and non-cooperative approaches using game theory. Through exploring the implementation of these techniques in popular technologies such as Maglev and Hadoop, the review highlights the advantages and limitations of each method in terms of system efficiency and fault tolerance. This study underscores the importance of carefully considering the trade-offs of different load balancing techniques to optimize system performance at low cost, maintain system availability, and ensure fault tolerance.

References

[1] Ali M Alakeel. A guide to dynamic load balancing in distributed computer systems. *International Journal of Computer Science and Network Security*, 10(6):153–160, Jun 2010.

[2] Shailendra S. Aote and M. U. Kharat. A game-theoretic model for dynamic load balancing in distributed systems. New York, NY, USA, 2009. Association for Computing Machinery.

[3] Sidra Aslam and Munam Ali Shah. Load balancing algorithms in cloud computing: A survey of modern techniques. In *2015 National Software Engineering Conference (NSEC)*, pages 30–35, 2015.

[4] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A fast and reliable software network load balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 523–535, Santa Clara, CA, March 2016. USENIX Association.

[5] D. Grosu, A.T. Chronopoulos, and Ming-Ying Leung. Load balancing in distributed systems: an approach using cooperative games. In *Proceedings 16th International Parallel and Distributed Processing Symposium*, 2002.

[6] Daniel Grosu and Anthony T. Chronopoulos. Noncooperative load balancing in distributed systems. *Journal of Parallel and Distributed Computing*, 65(9):1022–1034, 2005.

[7] Hesham A Hefny, Mohamed Helmy Khafagy, and M Wahdan Ahmed. Comparative study load balance algorithms for map reduce environment. *International Journal of Computer Applications*, 106(18):41–50, 2014.

[8] Einollah Jafarnejad Ghomi, Amir Masoud Rahmani, and Nooruldeen Nasih Qader. Load-balancing algorithms in cloud computing. *Journal of Network and Computer Applications*, 88:50–71, 2017.

[9] Mayanka Katyal and Atul Mishra. A comparative study of load balancing algorithms in cloud computing environment. *International Journal of Distributed and Cloud Computing*, 1(2), 2013.

[10] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma. Performance analysis of load balancing algorithms. *International Journal of Civil and Environmental Engineering*, 2(2):367–370, 2008.

Note: ChatGPT was used in the process of converting written notes into sentences for inspiring writing.