Scientific Research Publishing

International Journal of Internet and Distributed Systems, 2020, 4, 1-13
https://www.scirp.org/journal/ijids
ISSN Online: 2327-7165
ISSN Print: 2327-7157

Challenges and Considerations in Developing and Architecting Large-Scale Distributed Systems

Ion-Alexandru Secara

Human Factors and Ergonomics Society, San Francisco, California, USA
Email: ion.alexandru.secara@gmail.com

How to cite this paper: Secara, I.-A. (2020) Challenges and Considerations in Developing and Architecting Large-Scale Distributed Systems. *International Journal of Internet and Distributed Systems*, **4**, 1-13.
https://doi.org/10.4236/ijids.2020.41001

Received: February 10, 2020
Accepted: February 25, 2020
Published: February 28, 2020

Abstract

This paper investigates large-scale distributed system design. It looks at features, main design considerations and provides the Netflix API, Cassandra and Oracle as examples of such systems. Moreover, the paper investigates the challenges of designing, developing, deploying, and maintaining such systems, in regard to the features presented. Finally, the paper discusses aspects of available solutions and current practices to challenges that large-scale distributed systems face.

Keywords

Distributed Systems, Concurrency, Large-Scale

1. Introduction to Distributed Systems

Computing has changed significantly since the 1970s. The introduction of computer networks has led to the development of distributed systems. A distributed system is a collection of independent computers that appear to the user as a single computer. Components located at networked computers, which could be separated physically due to different locations, communicate, and coordinate their actions. The coordinated aggregation of those distributed components and their afferent resources facilitate high scalability and access to a larger amount of computing power [1]. Previous research has investigated challenges and consideration in developing small to medium scale distributed systems. However, the recent widespread usage of various smart gadgets and the increasing availability of the internet have tremendously increased the speed of research and innovation in this area. One key aspect, that became crucial especially to many businesses, was developing large-scale distributed systems [2]—some of the previous

research does not focus on the challenges and considerations in developing large-scale distributed systems or in some cases, and the research is based on certain assumptions since developing and maintaining large-scale distributed systems require considerable amounts of resources [3]. This led companies to develop their own distributed systems architectures and practices that scaled most efficiently for their business needs and quickly increasing usage load. One such example is Netflix, the 21st most accessed online platform in the world [4]. Netflix developed its own proprietary system, Ribbon, of load-balancers and DBSCAN, a fault tolerance monitoring tool, presented later in this paper. Those innovations as well as other advancements in the field have made it possible to provide new environments for data sharing, resource allocation, cycle sharing and other ways of interaction that involve distributed resources [5]. Hence, it became possible to relocate production units to decentralized zones and develop seamless large-scale distributed systems.

2. Main Design Considerations and Motives of Use

A distributed system is comprised of several nodes that are connected across a network. Software coordinates multiple nodes across a network, through messages. The coordinated aggregation of resources allows components to cooperate together to perform related tasks [1].

The nodes may have various roles within a distributed system. The role depends on the node's hardware specifications and software properties, as well as the system architecture (*i.e.* master-slave architecture) [6].

Distributed systems are provided by a variety of vendors. Thus, they use a variety of software components, based on each vendor's standards. Those systems are independent from the underlying software, in that they can run on various operating systems (individual nodes may operate Linux, Windows etc.). Also, the nodes can use various communication protocols—sets of rules to encode and decode the messages passed. Communication protocols can range from SNA, TCP/IP, Ethernet to Token Ring. For instance, HTTP (Hypertext Transfer Protocol) is the protocol for transferring messages over the World Wide Web [7].

In a standalone system, performance, storage, and many other features (scalability etc.) are limited to the hardware capabilities of the particular system. Centralized systems often experience high latency, thus poor overall performance, since parallel processing is limited and the physical distance from the majority of the users to the servers can be very large [8]. Large-scale enterprises require scalability as well as high performance levels to ensure a high Quality of Service (QoS) standard.

One of the main reasons of using a distributed system is sharing resources (software and hardware components) among clients and activities. Hence, distributed systems process concurrent activities in parallel, providing enhanced performance [9].

The resources are distributed at various geographical locations. Users can

connect to a data center that is physically closest, reducing latency. Moreover, in case a failure occurs within a data center, the other data centers can possibly take over the traffic, facilitating high availability.

For instance, Netflix Ribbon provides software load-balancers. It allows the client to communicate to individual servers, by supplying the IP or public DNS names of the servers. Groups of servers are divided into zones. Users have their address mapped and are redirected to the closest zone, to reduce latency. Ribbon also keeps track of statistics of zones. In case of high loads or latency to the existing nearest zone, users will get redirected to the next closest zone [10]. This improves overall QoS and reduces communication latency across the network channel.

3. Features of Large-Scale Distributed Systems

There are lots of issues that can arise when designing a distributed system. What if users request the same resources at the same time? What if one of the nodes undergoes a failure? Can a distributed system scale on demand?

Therefore, a distributed system has a multitude of defining features and characteristics. Those include concurrency, fault tolerance, no global clock, consistency, resilience and scalability [6].

3.1. Concurrency

On a single sequential processor, programs can be broken down into smaller independent tasks. Thus, the execution of those tasks can be interleaved. This creates the illusion of concurrency [6].

Moreover, large-scale distributed systems provide resources that are shared by multiple users and activities. Generated processes and user interactions happen at random times. Therefore, there is a high probability that several parallel process will access the same set of resources concurrently [9]. Concurrent access to the same resources can leave a distributed system in an inconsistent state.

Large-scale distributed systems are accessed multiple times at any point in time. Executing processes sequentially limits throughput. A single process can be broken down into smaller tasks that can be executed independently [6]. This would substantially increase the process execution speed.

On top of this, concurrency hides latency. It is often the case that some of the tasks are blocked because of external resources that they must wait responses from (*i.e.* disk or network I/O operations) [6]. Concurrency allows tasks that are not being blocked, to make use of available shared resources.

3.2. Challenges and Current Practices to Achieving Concurrency

Concurrency is affected by the extent to which tasks within a process need to interact with other processes, in order to successfully complete their execution. Also, there are processes that cannot be broken down into smaller tasks. Thus, they have to be executed sequentially [9].

Therefore, the challenge of achieving concurrency can be broken down into two separate issues.

As mentioned above, processes can be either executed as a whole or they can be broken down into smaller tasks that can be executed in parallel.

The resource management service, comprised of scheduling algorithms and the QoS agent, is used to allocate system resources to balance the loads of work and facilitate an optimal and efficient parallel execution of concurrent processes [9].

To prevent inconsistency, distributed systems use a form of process synchronization. This involves processors communicating with each other, using messages. Thus, processors can exchange changes in data and can adjust rate of process execution, in order to facilitate synchronization [11]. Once all the processes are finished, the different results are put together, to successfully finalize the execution.

Hadoop MapReduce is a framework that facilitates parallel execution, on multiple nodes, of processes involving huge amounts (multi-terabyte) of data. Each node runs the MapReduce framework and the Hadoop Distributed File system, to efficiently schedule tasks across multiple nodes and create a high bandwidth aggregation of data [12].

Hadoop MapReduce consists of two activities that allow parallel processing. The mapper creates intermediate key-value pairs out of the input, while the reducer takes the intermediate pairs and reduces them to unique keys and their respective total value. This is effective across many fields but is used in particular with tasks that process vast amounts of data [12]. A common use of the MapReduce is counting the frequency of words across multiple documents. This can be useful in algorithms involving natural language processing (**Figure 1**).

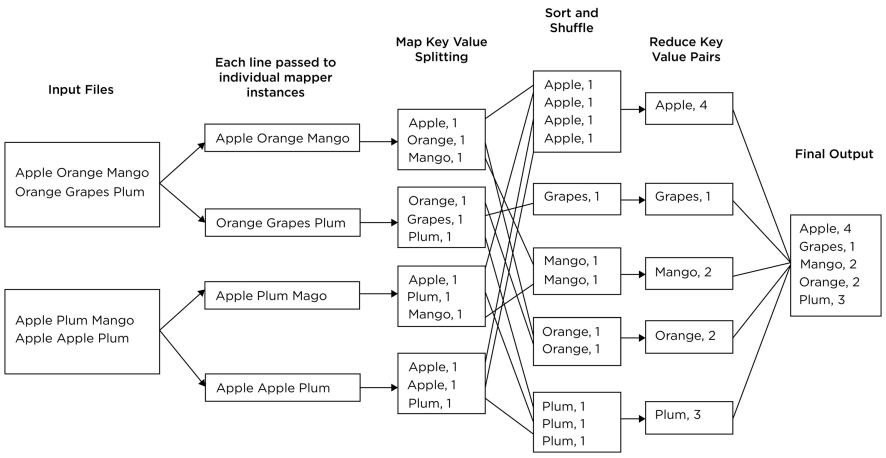

Figure 1. [13] Two input files with small samples of words are provided. The mapper splits the files further to execute multiple processes in parallel. Words within each block of data are mapped to key-value groups, representing the word and its current frequency. The reducer then sorts and shuffles keys together and reduces the values with the same key. The results process on different nodes is then grouped together into a single response.

Symmetric configuration is a popular configuration to execute concurrent processes in parallel. Each node has the same processor type and the same scheduling algorithm. Scheduling is decentralized—the resource management service holds a table listing of processes and their status [11].

Due to similar performance features, tasks are assigned based on the load of the respective node. The job can be broken down into smaller tasks that can be executed in parallel using multiple processors on the available nodes. This creates the need of process synchronization, which can create deadlock issues [11].

Netflix makes use of Apache Cassandra, as a Distributed Database Management System (DDBMS) [14]. Cassandra is an open-source distributed database system that offers high performance, availability and scalability, by replicating data and creating clusters of identical nodes across data centers—symmetric configuration [15].

Netflix employs a video encoding pipeline, across its EC2 instances. Jobs are divided into smaller tasks and executed in parallel. The internal spot market dynamically allocates jobs to nodes, based on real time availability of the computer resources [16].

Videos must be encoded in various quality representations and codec profiles, due to various viewing devices and levels of network connectivity; thus, the quality of a video has to adjust accordingly [16] (**Figure 2**).

Multiple processes can access and alter the same set of resources at the same time, within a large-scale distributed system.

Concurrency and data consistency are crucial in transactions. Transactions conform to the ACID standards (atomicity, consistency, isolation and durability). In order to ensure consistency, a schedule orders the execution of transactions [17].

In general, distributed systems enforce either locking protocols or the Multi-Version Concurrency Control technique.

Locking protocols are sets of rules that transactions follow when requesting

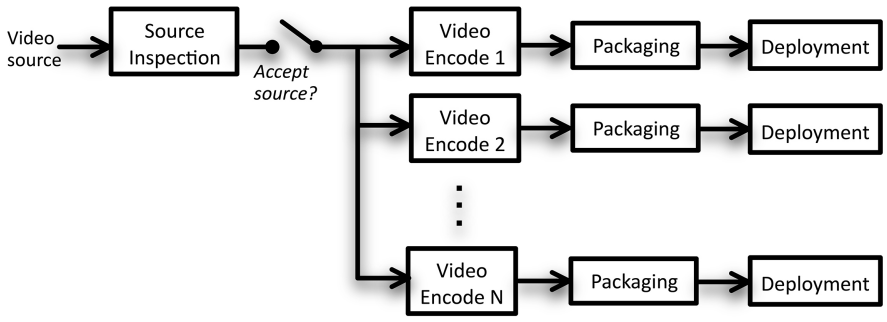

Figure 2. This shows a high-level overview of the video encoding processes. A single video source encoding is broken down into multiple tasks that are run in parallel on multiple processors. Each video source is further broken down into several chunks assigned to different nodes. After all the tasks are finished, the different chunks are put together and quality checked against the original video [16].

and releasing locks to the concurrency-control manager, in order to access a resource. A process must make a lock request to a resource, before accessing the actual resource [17].

A lock on a resource can be marked as exclusive (x)—the resource can be read and altered or shared (s)—the resource can only be read

A popular locking protocol is the Two-Phase Locking protocol. It is comprised of two phases, the growing phase (obtain and don't release locks) and the shrinking phase (release and don't obtain locks) [17].

The following scenario depicts, the concept of the Two-Phase Locking protocol (**Figure 3**).

The main challenge is running into deadlock – reader and writer locks are set to the same resources by different transactions. Thus, neither of the transaction can proceed execution.

To detect a deadlock, the systems set timestamps for each lock. This can tell whether a transaction is blocked. In case, the process runs into a deadlock, the system performs a roll-back—a transaction will be rolled back to a previous state [17].

MVCC is used by many enterprises, including Oracle, HyPer and Microsoft's Hekaton [18]. Before MVCC, locking protocols were the only viable option.

MVCC holds a snapshot of data (a database version) at a single point in time. The snapshot holds data as it was, before other currently uncommitted transactions were started [19]. Therefore, the transaction cannot view inconsistent data caused by other concurrent transaction updates, providing isolation.

Figure 3. As seen above, the execution of transactions is serial, in regards to their lock points. In the growing phase, both transaction acquire locks (Lock-X(b), Lock-X(c), Lock-X(a)), while in the shrinking phase they release the locks (Unlock(b), Unlock(c), Unlock(a)) [17].

Oracle uses the MVCC mechanism. Queries are provided read consistency, since all the data, the query sees, comes from a single point in time ("statement-level read consistency") [19].

Oracle uses roll-back segments that hold data values before it was changed by uncommitted or recently committed transactions. Data inside roll-back segments can be retrieved upon execution of read queries [20] (**Figure 4**).

Therefore, the query reads data with respect to the time when its execution began. Changes to data that occur during or after a query's execution start are not recorded. This guarantees that each query can access a consistent state of data.

3.3. Fault Tolerance

Multiple failures can occur within a distributed system. Those failures may not be detected immediately but can cause sudden changes in the performance of the system. The availability of the system measures the proportion of time that a system is available for use [5]. Hadzilacos and Toueg categorize failures into omission failures, arbitrary failures and timing failures, distinguishing between failures of processes and communication channels. Failure models such as this help categorize the occurrence of failures and the effects that they will have [6].

- Omission failures refer to failures that can occur within a processes or communication channel [6].

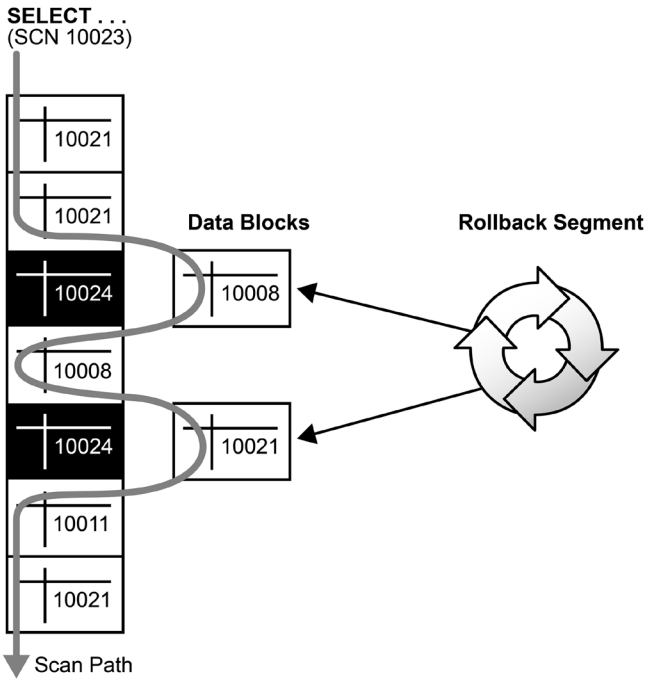

Figure 4. When a query is executed, is it given a current system change number (SCN), timestamp, "10023" in our case [20]. As the query proceeds through the available blocks, data with more recent (higher) SCNs is retrieved from the roll-back segment [20]. As seen above, the two "10024" blocks have their data retrieved from the rollback segment, since the SCN of "10024" is more recent that the query SCN "10023".

- A process can crash unexpectedly, due to a software error or a hardware failure. If the process does not answer to invocation messages sent by other processes, within a time-out period; the process is halted and will not perform its actions further. This is called a fail-stop. Depending on the design of the system, other running services can remain active if the tasks they execute can still function correctly [6].
- Communication omission failures occur when a message is not successfully transmitted by the outgoing message buffer to the incoming message buffer of another message, across the network channel. This can be due to lack of space on the incoming message buffer or by a network transmission error. This can leave nodes isolated, unable to proceed with the execution of their tasks [6].
- Arbitrary failures refer to failures that happen due to incorrect processing, rather than actual software or hardware crashes. They usually occur when a tasks sets incorrect values or returns wrong values when executed. This is caused by the logic of the process or randomly omitting tasks and instructions. Thus, corrupt message content or replicated messages can be sent over the communication channel [6]. However, those messages carry checksums and sequence numbers. The latter can detect non-existent or duplicated messages, while the earlier is used to detect corrupted message contents [6].
- Timing failures refer to failures that involve the process execution time passing the time-out limit. Thus, they can occur within synchronous distributed systems, since they have time-outs set on execution of processes. In asynchronous systems, no guarantees are offered on process execution time, thus timing failures are not really applicable [6]. A defect hardware component causing slow processing or slow message transmission speed can cause a timing failure. Moreover, processing and transmitting large pieces of data (*i.e.* videos, audio, images etc.) can require a considerable amount of time [6].

Large-scale distributed systems can consist of thousands of components. Although components may fail regularly, it is important that the system will keep running optimally.

Google, the world's most popular search engine, is accessed daily by millions of users. Some few usual failures that occur within the first year of a Google cluster of machines include [21]:

- approx. 1 PDU failure, which make 500 to 1000 machines disappear of the network channel. This requires between 1 and 6 hours to recover.
- approx. 1000 individual machine failures
- approx. thousands of hard drive failures

Those are only a few of the failures that happen. Therefore, system designers have to take into consideration any possible failures when designing a distributed system, since it is desirable for the distributed system to carry out its activity regardless of failures that occur.

The Netflix API receives more than 1 billion calls/day and sends out several

billion calls to underlying dependencies [22].

Imagine there are 30 dependencies with 99.99% uptime each. In a month, the dependency would have a downtime of 0.0001 × 30 = 0.003 or 0.3%. This results in an overall 99.7% uptime (100% - 0.3%), resulting in 2+ hours of downtime in a month. A single API dependency fail, can rapidly overflow all available Tomcat request threads, and crash the whole API [23].

Therefore, as for Netflix, a large-scale distributed system has to tolerate failures and continue its activity seamlessly.

3.4. Challenges and Current Practices to Achieving Fault Tolerance

Many failures are time-consuming or almost impossible to detect. A common practice is having a monitoring service that sets up agents at each of the four software levels—application, middleware, OS and network [6].

The Falcon spy network sets up "spies" at each of those levels. The network of "spies" sends back performance metrics of components at each layer and alert the system of any abnormal activity [24]. A simple architecture model of the Falcon Spy Network can be seen below.

Netflix uses an automated outlier detection service. It is known as the Density-Based Spatial Clustering of Applications with Noise (DBSCAN)—a cluster analysis technique, using unsupervised machine learning. A server's network connection can be very slow or defect, causing latency, or its system-level metrics can show abnormal behavior [23].

The effects of an unhealthy server may not overpass the default performance threshold of a distributed system, which does not allow fault detection but affects the customer service. This can be detected by a default threshold, but the server may be slow only temporarily, due to various reasons (*i.e.* high traffic rate) [23].

Netflix's DBSCAN groups nodes in clusters, such that nodes in the same cluster have similar traffic density and performance levels [25]. After grouping clusters together, DBSCAN marks outliers—deviating servers. Metrics to be monitored are collected from Atlas (primary time-series telemetry platform) and passed to DBSCAN [23].

Using machine learning, the performance of the outliers is evaluated against the performance of the cluster. Thus, a deviating server can be labelled as an outlier. The alerting service then alerts the system and can even terminate the server, allowing the auto-scaling group to replace [23].

3.5. Recovering from Failures

A failure can leave the system in an inconsistent state—unavailable services, lost data etc. Therefore, a common process is a roll-back recovery, which has two approaches—operation-based and state-based recovery [26].

In an operation-based recovery, all the modifications made to the distributed

system, over a certain period of time, are stored in log files [26]. Therefore, in case the system would fail at any point in time, the previous state of the system can be restored by reversing all the changes. The log files can be stored remotely, on other availability zones or using multiple storage providers. This would ensure to a certain extent, that they will be highly available and ready to be accessed in case of a large-scale failure (**Figure 5**).

A state-based recovery involves replication of data—periodic checkpoints or back-ups of the distributed system [26]. Back-ups can be stored in different availability zones, such that it can be made available at any moment. User requests can be redirected to the backup state of the system, while processes that were being executed are simply dropped [27].

Priam is a process that runs next to Cassandra on each node. It provides backup, recovery and metrics monitoring. During low traffic hours, a daily snapshot and modifications to the state of the distributed system are backed up on S3 (part of Amazon Web Services), offering an alternative backup strategy than the one employed by Cassandra [28].

During recovery, the Priam process on each node, downloads the respective snapshot and starts the cluster again. It restores the cluster to half the original size, by skipping alternate nodes and executing the repair process which reproduces the skipped data. This allows for a faster recovery—while Priam restores some of the data, the cluster can start producing the missing data [28].

However, to restart the cluster to a consistent state requires downtime. Therefore, Netflix employs another technique to achieve fault tolerance.

Netflix employs stateless services. Keeping copies of data and multiple running instances across various zones allowed Netflix to keep running (even during the AWS US-East outage). Netflix is designed for "N + 1" redundancy. This means that the platform allocates more capacity than it requires at any point in time. Thus, this surplus in capacity can support up to an entire AWS zone failure. When the availability zone failed, requests were redirected to other availability zones [28].

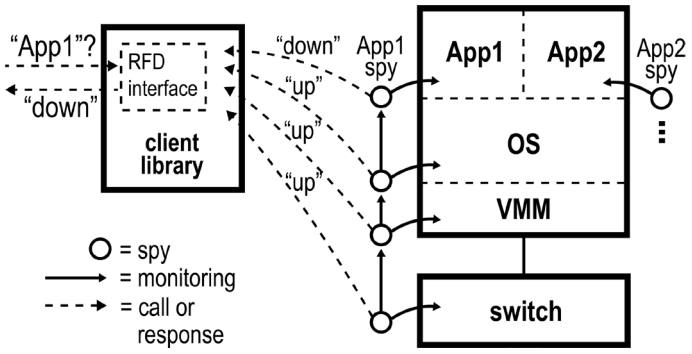

Figure 5. (Leners, Wu, Hung, Aguilera, & Walfish, 2011): Network of monitoring agents within the Falcon spy network. A "spy" lies at each software level and monitors the components within the specific layer.

4. Conclusion

In conclusion, there are many reasons that large-scale distributed systems have gained popularity over standalone systems. They offer benefits to clients, including a high QoS standard, by reducing latency and executing processes concurrently on multiple nodes. Despite facing many issues that can arise, distributed systems have developed solutions and current practices, such as Netflix's main design considerations and technology stacks that achieve, to a certain extent, fault tolerance, concurrency, and high availability. Therefore, distributed technologies will continue to improve and develop new technologies to achieve even higher standards.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Hajibaba, M. and Gorgin, S. (2014) A Review on Modern Distributed Computing Paradigms: Cloud Computing, Jungle Computing and Fog Computing. *Journal of Computing and Information Technology*, **22**, 69-84. https://doi.org/10.2498/cit.1002381

[2] Bagchi, S. (2015) Emerging Research in Cloud Distributed Computing Systems. IGI Global, USA, 158-166. https://doi.org/10.4018/978-1-4666-8213-9

[3] Hierons, R. and Nunez, M. (2010) Testing Probabilistic Distributed Systems. In: Hatcliff, J. and Zucca, E., Eds., *Formal Techniques for Distributed Systems. FMOODS* 2010, *FORTE* 2010. *Lecture Notes in Computer Science*, Vol. 6117, Springer, Berlin, Heidelberg, 63-77. https://doi.org/10.1007/978-3-642-13464-7_6

[4] Amazon (n.d.) netflix.com Traffic Statistics. Alexa. https://www.alexa.com/siteinfo/netflix.com

[5] Ahmed, W. and Wu, Y. (2013) A Survey on Reliability in Distributed Systems. Department of Computer Science and Technology, Tsinghua University, Beijing.

[6] Colouris, G., Dollimore, J. and Kindberg, T. (2005) Distributed Systems Concepts and Design. Addison Wesley, Boston, MA.

[7] IBM (n.d.) What Is Distributed Computing. IBM Knowledge Center. https://www.ibm.com/support/knowledgecenter/en/SSAL2T_8.2.0/com.ibm.cics.tx.doc/concepts/c_wht_is_distd_comptg.html

[8] Microsoft (2005, April 29) Centralized vs. Distributed Messaging System. TechNet. https://technet.microsoft.com/en-us/library/bb123575(v=exchg.65).aspx

[9] Hussain, H., Malik, S., Hameed, A., Khan, S., Bickler, G., Min-Allah, N. and Rayes, A. (2013) Parallel Computing.

[10] Netflix (2014, January 6) Working with Load Balancers. GitHub. https://github.com/Netflix/ribbon/wiki/Working-with-load-balancers

[11] Goodwin, D. (2013) Lecture #9: Concurrent Processes. Operating Systems. https://warwick.ac.uk/fac/sci/physics/research/condensedmatt/imr_cdt/students/david_goodwin/teaching/operating_systems/l9_concurrentprocesses2013.pdf

[12] Apache (n.d.) MapReduce Tutorial. Apache Hadoop.

https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[13] Bejoy, K.S. (2011, April 29) Word Count—Hadoop Map Reduce Example. Kick Start Hadoop. http://kickstarthadoop.blogspot.com/2011/04/word-count-hadoop-map-reduce-example.html

[14] Walz, E. (2016, July 7) How Netflix Uses a Distributed Database Management System to Deliver Your Movies. LinkedIn. https://phantom448.wordpress.com/2016/07/14/how-netflix-uses-a-distributed-database-management-system-to-deliver-your-movies/

[15] Apache Cassandra (n.d.) What Is Cassandra? Apache Cassandra. http://cassandra.apache.org/

[16] Netflix Technology Blog (2015, December 9) High Quality Video Encoding at Scale. The Netflix Tech Blog. https://medium.com/netflix-techblog/high-quality-video-encoding-atscale-d159db052746

[17] Letham, D.R. (2017, November 27) CS3101 Databases Lecture 19: Transactions. Studres. https://studres.cs.st-andrews.ac.uk/CS3101/Lectures/L19_Transactions.pdf

[18] Diaconu, C., Freedman, C., Ismert, E., Larson, P.-A., Mittal, P., Stonecipher, R. and Zwilling, M. (2015) Hekaton: SQL Server's Memory-Optimized OLTP Engine. Microsoft.

[19] Oracle (n.d.) Using Multiversion Concurrency Control Chapter 3. Berkeley DB Features. Oracle Docs. https://docs.oracle.com/cd/E17276_01/html/bdb-sql/mvcc.html

[20] Oracle (n.d.) Database Concepts. Oracle Help Center. https://docs.oracle.com/cd/B19306_01/server.102/b14220/consist.htm

[21] Dean, J. (n.d.) Software Engineering Advice from Building Large-Scale Distributed Systems. Google User Content. https://static.googleusercontent.com/media/research.google.com/en//people/jeff/stanford-295-talk.pdf

[22] Netflix Technology Blog (2012, February 29) Fault Tolerance in a High Volume, Distributed System. The Netflix Tech Blog. https://medium.com/netflix-techblog/fault-tolerance-in-a-highvolume-distributed-system-91ab4faae74a

[23] Netflix Technology Blog (2015, July 14) Tracking down the Villains: Outlier Detection at Netflix. The Netflix Tech Blog. https://medium.com/netflix-techblog/tracking-downthe-villains-outlier-detection-at-netflix-40360b31732

[24] Leners, J., Wu, H., Hung, W.-L., Aguilera, M. and Walfish, M. (2011) Detecting Failures in Distributed Systems with the Falcon Spy Network. *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, Cascais, October 2011, 279-294. https://doi.org/10.1145/2043556.2043583

[25] Harris, N. (2015, January 24) Visualizing DBSCAN Clustering. Naftali Harris. https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

[26] Yu, D. (2010, March) Recovery and Fault Tolerance. CSE 660 Operating Systems Concepts & Theory. http://cse.csusb.edu/tongyu/courses/cs660/notes/recovery.php

[27] Netflix Technology Blog (2011, April 29) Lessons Netflix Learned from the AWS

Outage. The Netflix Tech Blog.
https://medium.com/netflix-techblog/lessons-netflix-learnedfrom-the-aws-outage-deefe5fd0c04

[28] Sadhu, P., Parthasarathy, V. and Jami, A. (2012, February 21) Announcing Priam. The Netflix Tech Blog.
https://medium.com/netflix-techblog/announcing-priam-4165565c7b07