APACHE HADOOP PERFORMANCE EVALUATION WITH RESOURCES MONITORING TOOLS, AND PARAMETERS OPTIMIZATION: IOT EMERGING DEMAND

MO'TAZ AL-HAMI[1], MAJDI MAABREH[1], SALAH TAAMNEH[2], AAKASH PRADEEP[3], HANI BANI SALAMEH[4]

[1]Department of Computer Information Systems, The Hashemite University, Zarqa, Jordan 13115
[2]Department of Computer Science and Applications, The Hashemite University, Zarqa, Jordan 13115
[3]Salesforce, San Francisco, California, USA
[4]Department of Software Engineering, The Hashemite University, Zarqa, Jordan 13115

ABSTRACT

Recently, IoT has revealed a key value in the smart cities. Our living comfortability level has been improved. Such technology requires extensive data processing especially when it is a real time driven data. Apache Hadoop framework is a necessary and efficient model that can be incorporated with the IoT technology. Hadoop, the open-source framework, is typically used for off-line batch processing on large-scale clusters. It has a wide range of applications in the big data industry due to its capability in processing massive data in distributed and parallel environments. However, several aspects should be carefully evaluated before deploying Hadoop-based solutions. The authors thoroughly investigate the Apache Hadoop framework with the focus on factors that directly affect its performance. The work discusses and evaluates two crucial dimensions of Hadoop systems; monitoring tools and their impact on the performance of the Apache Hadoop based clusters, and the most influential parameters and the optimization techniques of Apache Hadoop based systems. Results showed that monitoring tools play a major role in Hadoop-based solutions planning and maintenance. According to the used experimental settings, the Cacti monitoring tool consumes around 45% of the memory usage, however memory usage in Ganglia is more efficient than Cacti tool (i.e., on average around 2.5%). For CPU utilization, both monitoring tools are efficient and the monitoring tool usage amount is almost negligible. The results also showed that there is a shortlist of critical parameters that significantly affect the overall performance. Based on the results, the authors conclude the paper by future directions and possible improvements that need further explorations and experiments.

Keywords: *Big data, IoT, distributed system, high performance computing, artificial intelligence, smart sensors.*

1. INTRODUCTION

The new technology Internet of Things (IoT) is powering the future of digital data processing. Precisely, the amount of generated data between the interconnected devices and sensors is relatively high. The way of storing, processing, and getting insight from data is a major issue. Apache Hadoop is considered a vital choice to incorporate it with IoT technology. Incorporating the Apache Hadoop framework in the IoT technology can reshape the data processing at the devices level, and the smart cities applications level. In general, the smart city architecture consists of three main tiers as shown in Figure 1. The front tier includes both peripheral sensors and devices nodes. Usually, the common characteristics of all nodes in this tier are the limited space and processing capability. Specially, when we deal with embedded computing environment the resources are limited and constrained [1][2].

The second tier is the middle tier, and it represents the smart gateway controller. It has the ability to issue smart decisions regarding data collection, summarizing, and routing [3]. At the same time, it forms the bridge that controls the movement of data between the Apache Hadoop and front tier devices and sensors. The third and back-end tier is the

Apache Hadoop framework. The majority of storage management and data processing are executred in this tier. The scope of this paper is to focus on the Apache Hadoop framework in terms of performance.

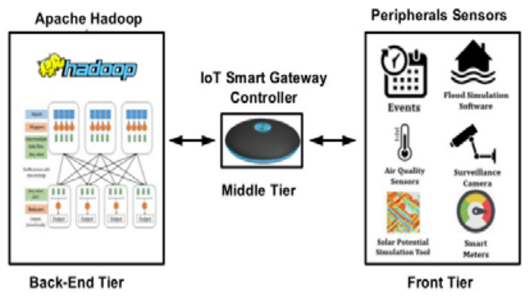


Figure 1: Example of a smart city architecture integrated with IoT technology. A smart gateway controller controls all the peripheral sensors.

Figure 2 is an efficient solution for processing large datasets. MapReduce has been studied and evaluated in terms of performance, scalability, and failure recovery in [4]. The work [5] discusses the MapReduce framework uses in Google search engine and its capability. Studying the trade-off between data locality a load balancing in MapReduce in order to maximize the throughput has been investigated in [6].
In MapReduce, the whole processing task is broken down into two stages: the map stage, and the reduce stage. At the beginning, data is divided into chunks and each chunk is assigned to a single mapper unit. Each mapper receives a chunk as an input and produces a value as an output. This process creates key-value pairs, which depend on the nature of the chunk content. Outputs from mappers apply further processing including sorting and grouping based on the key-value. The second stage is the reduce process. In this stage, the reducers combine and aggregate the results from mappers. Precisely, outputs resulted from sorting and grouping key-value pairs are assigned into different reducers. In each one, the reducer aggregates its input and produces a single output value. Several producers produce several outputs and store them in the HDFS.
Hadoop has the ability to handle large loads of datasets from different users in a consistent and efficient manner. Also, it can rearrange the usage of computational resources on the basis of the incurred load. Any enterprise uses Hadoop services can dynamically scale to any number of computing resources on the basis of traffic spikes and the changing environment.

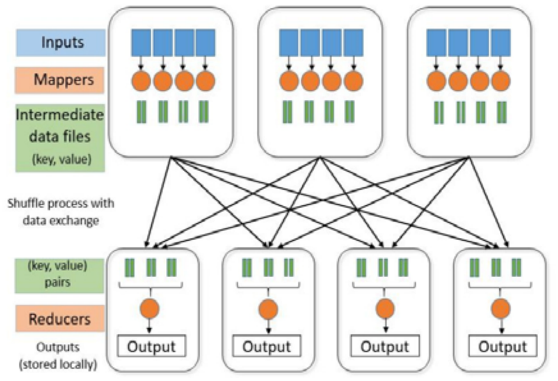


Figure 2: MapReduce framework. The tasks are divided into smaller chunks and used by mappers to produce key-value pairs. The reducers combine and aggregate results from mappers.

One of the main problems in the distributed environment is the hardware failure. Accessing a large number of hardware pieces increases the chance of failure that may happen among one of these hardware pieces. Hadoop Distributed File System (HDFS) is designed to store and manage a huge amount of data, running on a cluster on a commodity hardware. Since these commodity hardware resources maybe available from multiple vendors, the chance of a probable failure in any cluster node is probably high. The architecture design of the HDFS is constructed to manage and solve such interruption failures.

HDFS is highly fault-tolerant and is designed to be deployed on a low-cost hardware. HDFS provides a high throughput access to an application data and is suitable for applications that have large datasets [7]. Failure in HDFS has a nontrivial probability, which means that some components of HDFS might be non-functional. Therefore, the detection of any potential failures and the quick automatic recovery is a core architectural goal of HDFS. Monitoring HDFS is not a trivial task due to its large scale and distributed nature. At the same time, debugging Hadoop programs through monitoring logs is painful and impractical since it is excessively large.

Nowadays, there are many monitoring tools. These monitoring tools are effective and make the monitoring process simple. The main tasks of these tools are: first, extract HDFS metrics from the Hadoop logs. Second, correlate these metrics with the operating system level metrics such as CPU and memory utilization. This correlation is based on a per-job/per-task basis [7]. To clarify the system behavior, these tools collect the interaction of metrics information and visualize them in an interesting way, which increases the understanding level of the system behavior. Java Management Extensions (JMX) is a standard Java API for monitoring and managing applications. Different components of Hadoop: NameNode, DataNode, JobTracker, TaskTracker and RPC implement the JMX interfaces in order to report the data logs to JMX [8][9].

Monitoring the CPU utilization and the memory overhead is a key aspect of the performance [10][11]. The CPU utilization level affects the scalability of the whole cluster in terms of the number of used cores and the power consumption [10]. The work [11] studies the relationship of CPU load between host and guest machines under varying workloads conditions. Since the virtualization technology is gaining an increasing importance in cloud computing, the virtualization process has a strong relation with the infrastructure resources including CPU and memory usage amount [12]. Part of this work focuses on studying the amount of CPU utilization and the incurred memory amount while using such monitoring tools (i.e., Ganglia and Cacti). Since monitoring tools can add additional overhead, it is beneficial to take a look on the incurred overhead through using these tools. The nature of the architecture of each tool may affect the way used to monitor the cloud, and in turn the incurred overhead level. Such understanding of the monitoring performance can characterize the overhead amount, and the sources of that overhead.

Hadoop has been widely used for processing big data in a variety of domains. It is becoming a de facto for patch processing. In Hadoop, massive amount of collected data over a specific period of time is processed in a distributed fashion using the MadReduce programming model. The key reasons that make Apache Hadoop attractive for analyzing big data include: scalability, cost efficiency, flexibility, speed, and resilience to failures. Apache Hadoop has been adopted in many different disciplines. Hadoop has many applications in the field of the large-scale graph analysis. A subgraph analysis using Hadoop algorithm was developed by [13]. A new approach for graph management and analysis based on Hadoop was introduced in [14]. The approach uses a new set of operators for analyzing both single graph and collections of graphs. Moreover, a domain-specific language was proposed to define analytical workflows. Hadoop has also been used to compute the diameters of petabyte-scale graphs [15]. Additionally, several specialized distributed graph processing frameworks were built on top of the Hadoop, such as Giraph [16] and Hama [17].

Hadoop has also been used for medical big data. One study developed a Hadoop-based system to intelligently process medical big data and uncover some features of a hospital information system user's behavior [18]. Hadoop was intensively used for mining huge datasets. In clustering, a large number of Hadoop-based clustering algorithms have been proposed in order to reduce the execution time and to increase resilience [19][20][21]. The work [19] proposes a parallelize Genetic algorithm using MapReduce framework to apply clustering technique. In [20], parallel KMedoids algorithm is incorporated with MapReduce framework to breakdown the time complexity when dealing with big data. K-Means algorithm has also been used to cluster large datasets using MapReduce [21]. Several classification algorithms based on Hadoop, such as naïve bayes, nearest neighbor, and decision tree, have also been proposed [22][23]. C4.5 decision tree algorithm used the Hadoop framework to classify networks traffic [22]. In [23] satellite images are classified using the Hadoop framework. The aforementioned algorithms were mainly used for image classification, weather prediction, network traffic, and sentiment classification. Association rule learning has also been improved with the use of Hadoop. The apriori algorithm was among the most investigated algorithm in association rule learning [24][25]. The work [24] uses MapReduce Apriori algorithm based on Hadoop to find frequent pattern and apply data mining techniques.

Although it has been extensively used for analyzing data in a variety of domains, Hadoop has its own limitations when dealing with large data. The advantages and disadvantages of using Hadoop are summarized in **Table 1** for some of the domains using Hadoop intensively.

Table 1: The advantages and disadvantages of using Hadoop in a variety of domains.

Domain	Disadvantage	Advantages
Log analysis	**Slow processing speed:** The fact the Hadoop reads and writes the data from and to the disk for every stage	**Highly available:** Hadoop can continue functioning even if some of the NameNode nodes crash.

	of processing makes the whole process very slow.	**Highly scalable:** The number of nodes can be increased or de creased as needed. **Fault-tolerant:** Hadoop uses replication to ensure data availability even if any node fails. **Compatibility:** Hadoop is compatible with a wide range of big data tools. In fact, Hadoop is used as a data storage for many tools. **Cost-effective:** Hadoop is open source and can be installed on commodity machines.
Graph analysis	**Hadoop is not fit for small files:** Hadoop was designed to deal with small number of large files. When dealing with large number of small files Hadoop fails to provide satisfactory performance.	
Medical data	**Hadoop supports only batch processing:** This makes Hadoop inefficient for online processing needed in the medical field.	
Sentiment classification	**Hadoop is not secure enough:** to deal with sensitive data generated by social networks. Hadoop has no encryption or decryptions at the storage and network levels.	
Weather prediction	**Hadoop does not support real-time data processing.**	
Network traffic	**Hadoop does not support real-time data processing.**	
Bioinformatics data analysis	**Hadoop is not efficient for iterative processing.**	

The contributions in this paper are: First, the authors study the Apache Hadoop framework effect on resources through measuring the incurred overhead. Two monitoring tools were used (i.e., Ganglia and Cacti). Showing the performance of CPU and memory utilization while using these tools can improve the decision of choosing the monitoring tool in any cloud/cluster environment. Second, the authors study the optimization part of the Apache Hadoop that affects the framework performance. There are too many parameters control the framework performance. However, some parameters significantly impact the overall performance and they need to be evaluated carefully while deploying the Hadoop solutions.

In this study, the focus is distributed in two dimensions. The first dimension is to measure the Apache Hadoop framework performance. We measured the performance through two popular and widely used monitoring tools. The used monitoring tools are samples from a large number of available monitoring tools. The goal is to signify the importance of the choice of the used monitoring tool. Each monitoring tool has a different load on the used resources (i.e., CPU and memory). In case of using different monitoring tools, the analysis criteria would be the same. In the second dimension, the work study the major sensitive parameters that affect the Apache Hadoop performance. The best-used values that enhance the Apache Hadoop performance are analyzed and showed in an appropriate way. These parameters are very important for the overall performance based on the used cluster resources settings. Default parameters do not guarantee the best performance. The study shows that, the best-used parameters are different from the default settings.

The paper is organized as follows. The introduction appears in Section **Error! Reference source not found.**. Monitoring tools and Apache Hadoop optimization are studied in Section 2. The evaluation and analysis are described in 3. Conclusion remarks appear in Section 4.

2. MONITROING TOOLS

Ganglia is a scalable distributed monitoring system used to monitor several computing systems, such as clusters and grids. The Ganglia design depends on a hierarchical structure of federations of clusters (see Figure 3). The architecture of this hierarchical structure is implemented in a light weight overhead occurred in the cluster nodes, and at the same time assures high concurrency [26]. To visualize cluster metric results, Ganglia uses XML representation and RDtool for data storage and visualization. In addition, External Data Representation (XDR) format is used to serialize compact portable data transport. The architecture of Ganglia depends on multicast-based listen/announce protocol for monitoring process. To manage the membership inside a cluster, Ganglia uses a periodic heartbeat message on a predefined and announced multicast address. This periodic heartbeat message, identifies cluster members, so each node should receive a heartbeat messages from all other nodes in the cluster. At the same time, it reacts by sending a multicast heartbeat message to the other cluster nodes. In this way, cluster membership can be saved [26]. Figure 3 shows Ganglia architecture where leaf nodes specify cluster nodes, while the internal nodes specify aggregation points.

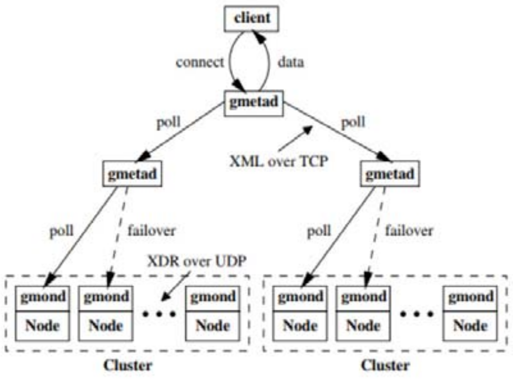


Figure 3: Ganglia tool architecture adopted from [26]. It uses gmond (i.e., Ganglia monitoring daemon) and gmetad (i.e., Ganglia monitoring daemon) daemons to monitor the cluster. leaf nodes specify cluster nodes, while the internal nodes specify aggregation points.

In Ganglia architecture, there are two daemons, gmond and gmetad, where each of these daemons has a specific task. At a single cluster level, gmond daemon provides a monitoring using listen/announce protocol, and it exists in every node. To respond to a client request, gmond uses an XML representation to retrieve the monitored data. On the other hand, Ganglia gmetad supports a federation between a group of clusters, where this allows monitoring across multiple clusters using a tree of TCP connections [26]. Since this work focuses on a single cluster, the authors support more details about single cluster monitoring using gmond.

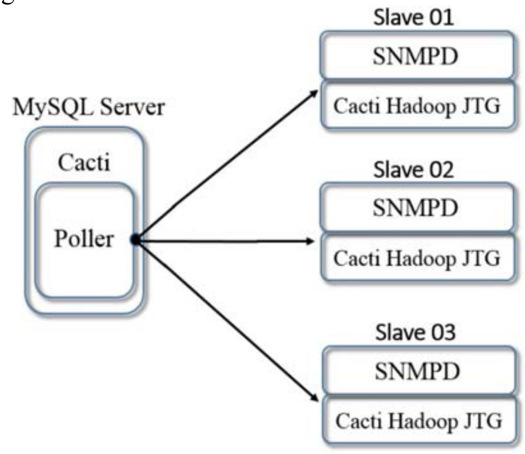


Figure 4: Cacti tool architecture. Cacti has three main operations: data retrieval, data storage, and data representation. Cacti uses a specific application executed at constant periods called poller, to collect data from the cluster nodes.

In gmond daemon, there are three different kinds of threads used to organize the monitoring process, and these threads are:

- **Collect and Publish Thread:** which is responsible on managing the node local state, so it collects the node information and publishes it for other nodes in the same cluster using a well-known multicast channel.
- **Listening Thread:** which is responsible on listening to the other nodes on the same cluster using a well-known multicast channel, and updates the node local memory state.
- **Export Thread:** which is responsible on managing and processing client's requests.

2.1 Cacti Monitoring Tool

Cacti is an open-source monitoring tool, which uses PHP/MySQL graphical representation. The design in Cacti depends on Simple Network Management Protocol (SNMP) that uses Round Robin Database Tool (RRD-Tool) engine to collect data. Cacti has three main operations: data retrieval, data storage, and data representation [27]. For data collection, Cacti uses a specific application executed at constant periods called poller, to collect data from the cluster nodes. Data fetched by Cacti is stored in RRD files. Regarding the tool management data, like user management and RRD files mapping, both are saved in MYSQL. RRDTool supports a customized graphical representation report to view the collected data from cluster nodes. Figure 4 shows the Cacti architecture.

2.2 Apache Hadoop Optimization

This section could be a good reference for newcomers and interested in the optimization problems in the Hadoop framework. It could be a good starting point to find the required research path that might link different dimensions together in more efficient and generic solutions especially for users of different technical backgrounds.

There are several issues need to be considered carefully when studying the performance of the Apache Hadoop framework. These issues include the cluster resources including the used CPUs power, and the used memory capacity. The shape of the used resources can control the used Apache Hadoop framework structure including the number of mappers and reducers used in the framework.

Next, the application characteristics and its relation to the available resources is an important issue has to be studied carefully. The effort of tuning Hadoop systems is mainly to find the best function, which can describe the relationship among the parameters, hardware, and application characteristics that maximize the capacity and system performance.

Generally speaking, computational systems are governed by a set of parameters that should be adjusted to end up by the desired capacity and performance. Sometimes, a parameter may play a major role in systems crash and failure by selecting either higher or lower values of what is needed. Hadoop has hundreds of interconnected parameters that need to be carefully studied before deploying Hadoop systems. The focus could be on those related to I/O, memory, and CPU utilization. Besides, given the size of data intended to be processed in Hadoop on a commodity hardware, finding the optimal values of Hadoop parameters is a challenge [28].

Hadoop provides different ways to facilitate the configurations to maximize the benefit of Hadoop power and capacity, and also to manage the different needs of applications and the available hardware. Parameters values can be set by updating the XML files of parameters (core-default.xml, hdfs-default.xml, hdfs-rbf-default.xml, mapred-default.xml, yarn-default.xml), using the command line, or by some lines of code using Configuration class [29].

The effort of tuning Hadoop systems is mainly to find the best function which can describe the relationship among the parameters, hardware, and application characteristics that maximize the capacity and system performance. Out of hundreds of parameters, few of them are the most influential on the overall performance [29][30] based on several empirical studies. The work [29] provides a customizable Apache Hadoop framework that allows the user to customize some functionality parameters that control resources like CPU and memory. summarizes the common parameters that have a considerable impact on Hadoop systems. In terms of different applications, there is a clear variation in parameter values from one to another and from dataset size to a larger or smaller one. This is one motivation of several studies to find the best recipe for Hadoop parameters to maximize the performance and resource utilization of the MapReduce application.

The contributions in Mapreduce parameters' tuning fall into five main categories; rule of thump, expert developers and engineers like [28], machine learning Models, cost-based models, and search-based solutions. However, profiling is the major process that paving the way for them. Another taxonomy is discussed in recent research [31] where the tuners of Hadoop come in six categories; rule-based, cost modeling, simulation-based, experiment-driven, machine learning, and adaptive tuning. One could see them in two categories in terms of their methodology as online such as [32] [33] or offline such as the solution [34]. The work [33] proposes an online approach for performance tuning which monitors a job's execution performance, and tunes associated performance-tuning parameters using some collected statistics. Starfish [35] is considered a very early attempt in Hadoop parameters tuning. The work has been built on the basis that the overall performance of the Hadoop-based (MapReduce) systems is an optimization problem of three main inputs; the configuration of job parameters, the resources allocated to run a job, and the properties of the dataset being processed. Given a large number of parameters in this optimization problem, Starfish focuses on some parameters where the empirical study highlighted some of the Hadoop parameters that have more effect on the performance. Besides, more complex interaction among the parameters where, for example, "io.sort.factor" showed a significant improvement in the execution time while "mapred.reduce.tasks" parameters assigned values less than 50, and almost no effect if "mapred.reduce.tasks" set to higher values, like 300. Another example is a recent experiment shows that optimizing concurrent tasks on the same node along with the block size parameter that shows significant improvements in the overall YARN system [36]. A recent experiment [37] applied a feature selection approach to find those more important parameters than others which could narrow down the list of parameters that need to be adjusted. The results highly conformed with the empirical evaluation in different studies and report "mapreduce.job.reduces", "mapreduce.job.maps", "mapred.child.java.opts", "mapreduce.task.io.sort.mb" may have the most impact on the execution time of the MapReduce process. The same parameters were reported of about 8 years before this study when [38] evaluate the MapReduce parameters using principal component analysis, PCA.

Starfish is a cost-based solution depends on measuring the CPU and I/O while profiling jobs. Machine learning-based solutions could be a good solution, however, the lack of large enough data could result in inefficient models, thus poor results and performance. The work [34] proposed and evaluates search-based solutions by Genetic algorithm. The adaptive solution can find semi-optimal parameters' values in less than 30 trials. Despite the encouraging results compared to Starfish and manual tuning, we need to evaluate this solution with larger datasets that are currently the challenge of many platforms and solutions. [39] is another cost-based solution that uses a search-based algorithm to find good configurations through sequential steps on profiling data.
One of the obstacles of applying the Starfish approach for ad-hoc systems is the properties or data distribution and statistics are not defined ahead of MapReduce processing. An efficient data sampling could add value to the overall solution by task distribution and load balancing. Compared to the Starfish tuner, [40] proposed a machine learning-based approach where support vector regression (SVR) was able to set more parameters in an adaptive solution which depends mainly on the observations of the systems on different applications. Datasets were collected from multiple experiments on different benchmarks of different sizes. Next, the sampling technique is used gradually to avoid random sampling and to quickly build a machine learning model (i.e., hypothesis) instead of using the whole dataset, which is relatively large.

Ant [32] is a search-based tool that starts tasks with random configurations and then gradually improves them while tasks are running. Ant shows improvement with large tasks and I/O intensive compared to itself with small and CPU intensive tasks. That is tuning the parameters needs some time and small jobs are usually done before the decision. Ant still works under the assumption that the tasks are long and have a uniform completion time on the same hardware. To handle a different case, [33] proposed MRONLINE, an online tool, which allows different configurations of different tasks instead of having the same values of parameters for all tasks.

Jellyfish [41], is an online tool, which still utilizes the cost-based and profiling methods in online tuning. The main idea is using the statistics collected from different tasks with different configurations and then decide the best setup. Jellyfish separates searching space into map-related parameters and reduce-related parameters, which, in turn, dramatically speedup the searching process. It also provides a resource re-schedule mechanism that maximizes the busy time of the resources.

Different machine learning methods such as clustering and classification are used along with the cost-based model in multi-phases tuners. [42] provides a self-tuning solution by applying the profiling and clustering methods on the applications. Mapreduce-based applications could be separated into different clusters based on their usage of CPU, I/O, and Memory. The second phase of the solution is mainly to identify the class the new job belongs to. This is by running the job on a small sample of data and then collect system performance data (CPU, I/O,...) where the classifier decides which values of parameters should be used that could maximize the performance.

RFHOC tool [43] consists of subsequent phases and also depends on profiling to collect data about MapReduce tasks execution times and configurations to feed the Random forest algorithm and then the Genetic algorithm in the next subsequent phase to find the best configuration. The speedups are significant compared to cost-based optimization on various sizes of datasets; from 50 GB to 1 TB. The training dataset is collected from running random different configurations on a subset of the original dataset being processed. The experiments also showed that generating the training dataset for random forest could need days, but to train a random forest model is typically a very fast task on small matrix; about 2000 records.

Linear regression is utilized in [44] to predict the execution time of MapReduce jobs. The model was trained on log files data and then the outcomes are consumed by a process to reduce the makespan of a job. This, in turn, reduces the overall execution time and also improves resource utilization. KNN algorithm was also used to train a model over a 10 months log history of MapReduce Yahoo data [45]. The main objectives are to find similar jobs in terms of resource needs and to build a model predicting the completion time. "Dstat" tool collects CPU and Disk usage data and represents the main source of training data to build support vector machine (SVM) which is the heart of AROMA tool [46]. Tree-based regression and ensemble methods are used to estimate the performance of MapReduce distributed jobs [47]. Random forest feature importance is also utilized in this solution to narrow down the parameters list. The solution reaches about more than 90\% of accuracy on Terasort and wordcount benchmarks algorithms.

Profiling and data collection like in [43] could be considered a time-consuming task and also needs resources, however, this overhead is not a frequent task since the built model could be used for future tasks and no need to roll the ball again. This case would be changed if the datasets being processed vary from time to time, thus the task to revisit the configurations becomes a frequent task which increases the overhead over the infrastructure. The interesting question would be about the role of machine learning and how much help can provide to predict the best configuration after a long time of profiling data so that minimizing the need to keep profiling the applications?

A recent study [48] showed that applications that belong to the same pattern or share pattern characteristics are likely to share also the performance characteristics on the same data size. This interesting result may guide the research to evaluate patterns-based solutions on big data. Machine learning models could have more accurate results and the need for continuous training/retraining the model might be minimized especially that another recent study [37] narrows down the list of important parameters to a shorter list.

3. EVALUATION

In the evaluation, the authors focus on studying Apache Hadoop effect on memory and CPU. For this purpose, two different monitoring tools Cacti and Ganglia were used. Both monitoring tools show memory and CPU status while running.

3.1 Experimental Setup

The experiment were executed on a local experimental on-premise HPC. It is used as a service private cloud geared towards scientific computing research. The used cloud includes two main parts: virtual machines, and elastic block storage. In this work, the authors created a cluster consists of three virtual machines. In these virtual machines, only a single CPU is used, the used memory size is 128 MB, and the disk space is 2 GB. Figure 5 shows the used cluster setup. For all the experiments, Hadoop-0.20 version and Puppet installation tool for it were used. During the experiments, the incurred overhead in the CPU and memory using the monitoring tools is measured.

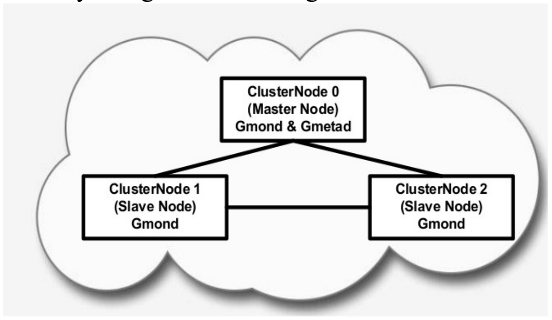
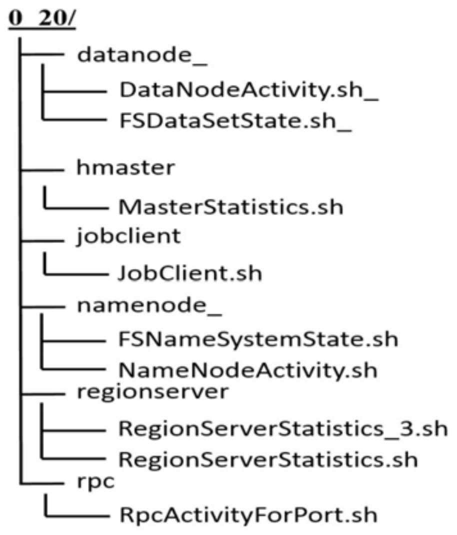


Figure 5: The used cluster setup. The authors created a cluster consists of three virtual machines. In these virtual machines, only a single CPU was used, the used memory size is 128 MB, and the disk space is 2 GB.

3.2 Cacti Analysis

Cacti monitoring tool uses a pulling approach for accessing metrics in the cluster nodes. Precisely, to read the measured CPU overhead and the incurred memory usage metrics, polling mechanism is used. The polling process is organized in regular intervals. In this work, the authors used the Cacti plugin Hadoop-Cacti-jtg, which is designed for graphing JMX attributes from Hadoop with Cacti. ***Figure 6:*** shows the scripts



```
0_20/
├── datanode_
│   ├── DataNodeActivity.sh_
│   └── FSDataSetState.sh_
├── hmaster
│   └── MasterStatistics.sh
├── jobclient
│   └── JobClient.sh
├── namenode_
│   ├── FSNameSystemState.sh
│   └── NameNodeActivity.sh
├── regionserver
│   ├── RegionServerStatistics_3.sh
│   └── RegionServerStatistics.sh
└── rpc
    └── RpcActivityForPort.sh
```

Figure 6: Scripts for polling data from Hadoop using Hadoop-Cacti-jtg (i.e., Cacti plugin for graphing JMX).

provided by the Hadoop-Cacti-jtg for polling data from Hadoop. For this purpose, the Cacti was configured to use these scripts to poll Hadoop data from the master and slave nodes. For example, to get the Namenode data FSNamesystemState.sh and NameNodeActivity.sh scripts were used [49].

In the used experiment setup, the polling process is applied to read the memory usage and the CPU overhead. For memory usage, the observed overhead is relatively high specially in the peak periods. Since the cluster is not performing any tasks other than Cacti monitoring tool, the incurred memory usage is related to the Cacti tool. Figure 15 shows the measured memory usage for four different types of daemons; Cacti, Apache HTTP server, SNMP, and MySQL. The polling process time is measured in seconds. The observed reads show that Cacti and Apache HTTP server consumes amount of memory around 45% in the peak periods, and this is relatively high. The memory usage by other daemons like SNMP and MySQL around 5%

which is relatively low and negligible. The observed CPU usage of the Cacti is 0% most of the time.

3.3 Ganglia Analysis

For Ganglia monitoring tool, three different scenarios were used to measure the CPU overhead and the memory usage in every node in the cluster:

- When Apache Hadoop framework is idle (i.e., not doing any kind of processing or file read/write operations). In this experiment, the authors tried to capture the absolute overhead of Ganglia.
- When Apache Hadoop framework executing Tera sort algorithm, which involves (1) generating the data using Teragen unit, (2) sorting the data using Tera sort and then (3) validating the sorted data. Here, the main goal is to capture the effect of preforming Apache Hadoop framework with Ganglia.

The aforementioned scenarios can measure the effect of Apache Hadoop and its overload, also they show the effect of Ganglia monitoring tool when using to monitor Apache Hadoop framework.

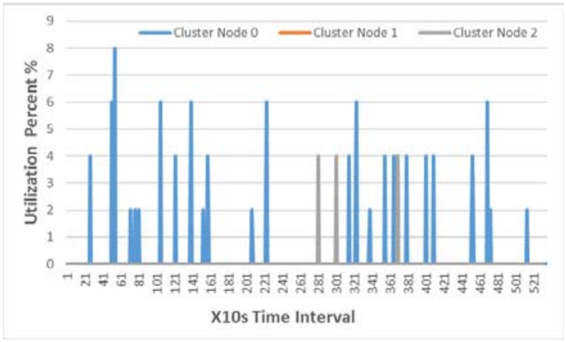


Figure 7: CPU utilization using gmond tool (i.e., Ganglia monitoring daemon) [10 seconds].

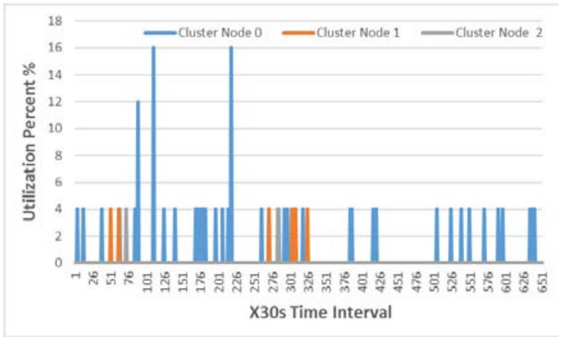


Figure 8: CPU utilization using gmond tool (i.e., Ganglia monitoring daemon) [30 seconds].

Polling mechanism in Ganglia has an effect on the incurred overhead. To show the effect of the polling process, three different intervals for polling process were used: 10 seconds, 30 seconds, and 60 seconds. Throughout the analysis, Cluster_node 0 refers to the master node (i.e., Hadoop server node) and it runs gmond and gmetad daemons at the same time. Cluster_node 1 and Cluster_node 2 refer to the Hadoop slave nodes and they run gmond daemon only. In Figure 3, the daemon gmond is located at the leafs level, were slave nodes reside there. The polling process at this level follows XDR over UDP transmission protocol, were in case of any potential failure slave nodes do not react to the polling process. For the master node, were Hadoop server resides, XML over TCP transmission is included beside gmond daemon.

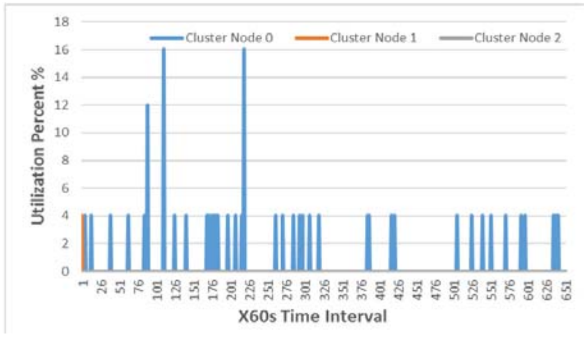


Figure 9: CPU utilization using gmond tool (i.e., Ganglia monitoring daemon) [60 seconds].

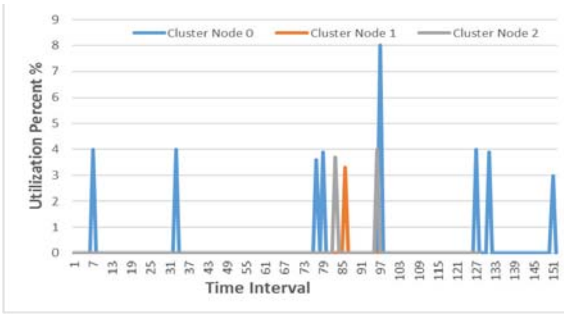


Figure 10: CPU utilization using gmond tool (i.e., Ganglia monitoring daemon) [Tera sort].

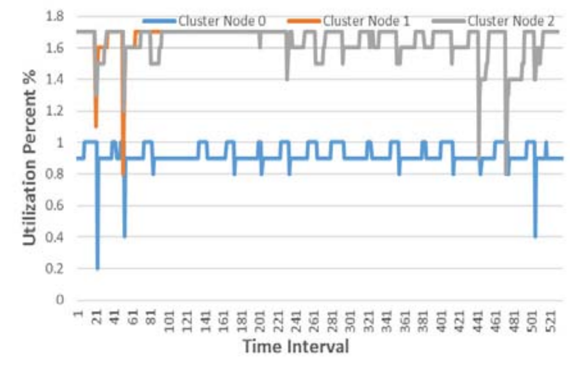


Figure 11: Memory utilization using gmond tool (i.e., Ganglia monitoring daemon) [10 seconds].

For the first scenario when Apache Hadoop is idle, the generated CPU overhead is negligible. In all different polling intervals, the CPU has no overhead, which means the polling process does not cause overhead (see Figure 7 - Figure 9). The observed difference between the master node Cluster_node 0 and the slave nodes Cluster_node 1, and Cluster_node 2 is almost negligible.

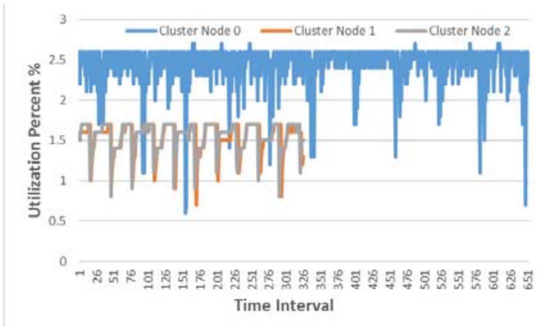


Figure 12: Memory utilization using gmond tool (i.e., Ganglia monitoring daemon) [30 seconds].

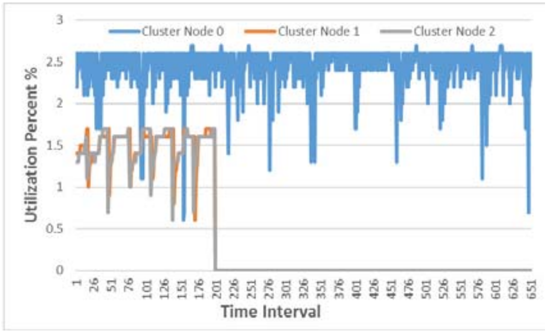


Figure 13: Memory utilization using gmond tool (i.e., Ganglia monitoring daemon) [60 seconds].

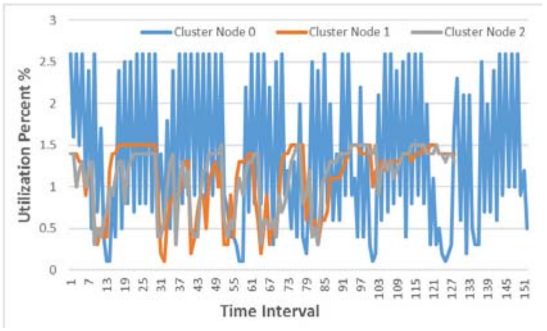


Figure 14: Memory utilization using gmond tool (i.e., Ganglia monitoring daemon) [Tera sort].

As a conclusion, the Ganglia monitoring tool is very efficient in the first scenario (i.e., when the Apache Hadoop framework is idle) and using different polling periods.

In the second scenario, when there is an application running (in our case Tera sort). In this Scenario, the Hadoop framework with MapReduce was able to elevate the incurred overhead in a way makes it negligible. The CPU deals mainly with two main parts, the Hadoop framework, and the Ganglia monitoring tool at the same time. ***Figure 10*** shows the generated overhead in the CPU for each node. The results show that the amount of overhead is slightly negligible and the Ganglia tool is a lightweight process in generating overhead. The master node Cluster_node 0 has a little more overhead since it is the Hadoop server node. As a conclusion from this experiment, the Apache Hadoop framework is efficient when dealing with the CPU. Regarding the memory utilization when using the Ganglia monitoring tool, the same experiment was repeated. ***Figure 11*** - ***Figure 14*** show the memory utilization for Ganglia monitoring tool. The same aforementioned scenarios were followed. When Apache Hadoop is idle, the memory utilization is shown in (***Figure 11*** - ***Figure 13***). For different polling periods the memory utilization in all the cluster nodes is negligible. Thus, the polling period has no effect on the memory utilization. In the second scenario, when using Tera sort (***Figure 14***), the memory utilization variation appears to be more than the scenario when Hadoop is idle. However, the memory utilization is still negligible in all the cluster nodes, making the ganglia tool as an efficient choice for all scenarios.

In (***Figure 16*** - **Figure 18**), a comparison between gmetad and gmond in Ganglia monitoring tool. The goal is to study the difference between the generated reads through using these different daemons. The same aforementioned scenarios when the Apache Hadoop is idle are followed. Each daemon has a different memory utilization and incurred CPU overhead. Using different polling periods (***Figure 16*** - **Figure 18**), the memory utilization level for both daemons is always larger than the incurred overhead in the CPU, however both of them are relatively low. As a conclusion, Apache Hadoop framework does not cause any heavy overhead in the CPU, however the memory utilization vary based on the used monitoring tool (i.e., Ganglia or Cacti) beside the Hadoop framework.

3.4 Parameters in Ganglia and Cacti

In the previous sections, the authors noticed that the monitoring tools Ganglia and Cacti each has a different architecture and polling strategy. The polling strategy has an effect on the memory utilization especially for Cacti were the poller is centralized in the Apache Hadoop server node. It is important to compare between the main available parameters in Ganglia and Cacti. ***Table 2*** shows the main parameters settings in Ganglia and Cacti. These parameters in general capture different monitoring aspects like CPU status, memory status, network status, visualizing process, and the type of the used database.

Table 2: Ganglia and Cacti Parameters Comparison.

Parameter	**Monitoring Tool**	
	Ganglia	Cacti
License	BSD	GNU
Default monitor with Hadoop	Default	Not Default
Web interface	Available	Available
Extensibility	Yes	Yes
User Management	No	Yes
RRD Tool	Yes	Yes
JMX	Yes	Yes
Hadoop data retrieval model	Push	Pull
Hadoop alert	Yes	Yes
XML based data transfer	Yes	No
CPU monitoring	Yes	Yes
Memory monitoring	Yes	Yes

3.5 Parameters and Optimization

The experiments were conducted using a virtual machine running on Google cloud, with 32 vCPUs and 256 GB of RAM. The main goal, behind the experiment in this section, is to show the effect of changing the default values of the parameters understudy on the overall performance. To achieve this goal, the common wordcount algorithm has been used on around a 6 GB dataset. The experiments show a significant improvement in the execution times using the same dataset but with different values of parameter a little far from the default values (see Table 3). Of course, the values could be different if another high specs cluster is used or a larger dataset is fed to the algorithm. The proposed values of the parameters range from the defaults and gradually increased till the execution time does not significantly change. Table 3 also shows that these parameters got attention in different studies and the range of tuned values, sometimes, relatively large.

Table 3: Parameters optimization for wordcount problem.

Parameter name	The experiment parameter value
dfs.replication	4
dfs.block.size	100, 128
mapreduce.task.io.sort.mb	100-128
mapreduce.task.io.sort.factor	200, 256
mapreduce.map.sort.spill.percent	0.7
mapreduce.job.reduces	16
mapreduce.job.maps	48
mapred.child.java.opts	512
mapreduce.reduce.shuffle. parallelcopies	8

4. CONCLUSION

The IoT technology has become a key technology in several applications including smart cities applications. However, an extensive data processing is required especially for real-time applications. Apache Hadoop is a core component framework necessary for processing such data. The authors thoroughly investigate the Apache Hadoop framework through studying the factors that directly affects the framework performance.

The work discusses and evaluates two crucial dimensions of Hadoop systems; monitoring tools and their impact on the performance of the Apache Hadoop based clusters, and the most influential parameters and the optimization techniques of Apache Hadoop based systems. The authors mainly discuss and evaluate two crucial aspects of Hadoop-based systems that might be risky in the sense that arbitrary decisions on them could be costly and/or distract users from their main projects goals. When talking about IoT technology, Apache Hadoop is one of the main things we need to consider. Resources monitoring and parameters optimizations show a significant impact on the overall performance of Hadoop systems. The authors focus their study on two commonly used monitoring tools of the Hadoop framework; Ganglia and Cacti, in a distributed environment. The focus was on CPU and memory utilization. One important observation, the result revealed that changing the frequency of polling time has no major effect on the monitoring process, and both tools can send useful feedback about the infrastructure status while running Hadoop using different polling time values. This observation could help the administrators in setting up Hadoop solutions where the resources usually busy in Hadoop processes.

The second dimension of the study showed that the Hadoop framework is sensitive for a few parameters, out of hundreds, where they need to be adjusted wisely. The authors found that previous works in Hadoop performance tuning can be one of five major approaches; the rules of thump, expert developers and engineers, machine learning Models, cost-based models, and search-based solutions. Some solutions may come in a hybrid model of the above. However, all of them need profiling data on different applications and environments.

Results showed that monitoring tools play a major role in Hadoop-based solutions planning and maintenance. The choice of the used monitoring tool has an effect on the available resources (i.e., CPU and memory). The results also showed that there is a shortlist of critical parameters that significantly affect the overall performance.

Future research could focus more on designing tools for cloud-based Hadoop solutions where data fusion techniques may help in collecting performance data from cloud resources (i.e. might be in different places). In terms of profiling for parameter optimization, the current process of profiling is time-consuming, especially on big data. One suggestion that needs further experiments is to cluster applications based on their design patterns or design characteristics. One more dimension is the employment of machine learning to analyze profiling data of different applications/environments to predict the best configuration. Given the increasing size of daily data, Hadoop monitoring, and performance tuning are major challenges that need extensive practical studies on cloud infrastructure.

ACKNOWLEDGMENT: Dr. Maabreh (the second author) acknowledges that the Deanship of Scientific Research at The Hashemite University has provided a financial support for partially funding this research.

REFERENCES

[1] M. Al-Hami, M. Pietron, R. Casas, M. Wielgosz, "Methodologies of compressing a stable performance convolutional neural networks in image classification," *Neural Processing Letters,* pp. 105--127, 2020.

[2] M. Al-Hami, M. Pietron, R. Casas, S. Hijazi, P. Kaul, "Towards a stable quantized convolutional neural networks: An embedded perspective," in Proceedings of the International Conference on Agents and Artificial Intelligence, Madeira, Portugal, 2018.

[3] V. Diaconita, A. Bologa, R. Bologa, "Hadoop oriented smart cities architecture," Sensors, vol. 18, no. 4, 2018.

[4] J. Dean, S. Ghemawat, "MapReduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107--113, 2008.

[5] J. Dean, S. Ghemawat, "MapReduce: a flexible data processing tool," Communications of the ACM, vol. 53, no. 1, pp. 72--77, 2010.

[6] W. Wang, K. Zhu, L. Ying, J. Tan, L. Zhang, "Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," IEEE/ACM Transactions on Networking (TON), vol. 24, no. 1, pp. 190--203, 2016.

[7] K. Shvachko, H. Kuang, S. Radia, R. Chansler, "The hadoop distributed file system," in IEEE 26th symposium on mass storage systems and technologies (MSST), 2010.

[8] J. Venner, Pro hadoop, Apress, 2009.

[9] T. White, Hadoop: The definitive guide, O'Reilly Media, Inc, 2012.

[10] M. Gusev, S. Ristov, M. Simjanoska, G. Velkoski, "Cpu utilization while scaling resources in the cloud," Cloud Computing, pp. 131--137, 2013.

[11] K. V. Chivukula, Monitoring and Analysis of CPU load relationships between Host and Guests in a Cloud Networking Infrastructure: An Empirical Study, 2015.

[12] N. Huber, Q. M. von Quast, M. Hauck, S. Kounev, "Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments.," in *CLOSER*, 2011.

[13] Z. Zhao, G. Wang, A. R. Butt, M. Khan, V. A. Kumar, M. V. Marathe, "Sahad: Subgraph analysis in massive networks using hadoop," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, 2012.

[14] A. P. K. G. E. R. M. Junghanns, "Gradoop: Scalable graph data management and analytics with hadoop," *arXiv preprint arXiv:1506.00548,* 2015.

[15] U. Kang, C. Tsourakakis, A. P. Appel, C. Faloutsos, J. Leskovec, "Hadi: Fast diameter estimation and mining in massive graphs with hadoop," *ACM Trasactions on Knowledge Discovery from Data (TKDD),* vol. 5, no. 2, 2008.

[16] C. Martella, R. Shaposhnik, D. Logothetis, S. Harenberg, Practical graph analytics with apache giraph, Springer, 2015.

[17] S. Seo, E. J. Yoon J, J. Kim, S. Jin, J. Kim, S. Maeng, "Hama: An efficient matrix computation with the mapreduce framework," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, 2010.

[18] Q. Yao, Y. Tian, P. Li, L. Tian, Y. Qian, J. Li, "Design and development of a medical big data processing system based on Hadoop," *Journal of medical systems,* vol. 39, no. 3, 2015.

[19] N. Hans, S. Mahajan, S. Omkar, "Big data clustering using genetic algorithm on hadoop mapreduce," *Int. J. Sci. Technol. Res*, vol. 4, pp. 58--62, 2015.

[20] Y. Jiang, J. Zhang, "Parallel K-Medoids clustering algorithm based on Hadoop," in 2014 IEEE 5th International Conference on Software Engineering and Service Science, 2014.

[21] C. Sreedhar, N. Kasiviswanath, P. C. Reddy, "Clustering large datasets using K-means modified inter and intra clustering (KM-I2C) in Hadoop," Journal of Big Data, vol. 4, no. 1, 2017.

[22] Z. Yuan, C. Wang, "An improved network traffic classification algorithm based on Hadoop decision tree," in 2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS), 2016.

[23] I. Chebbi, W. Boulila, I. R. Farah, "Improvement of satellite image classification: Approach based on Hadoop/MapReduce," in 2016 2nd

International Conference on Advanced Technologies for Signal and Image Processing (ATSIP), 2016.

[24] O. Yahya, O. Hegazy, E. Ezat, "An efficient implementation of a-priori algorithm based on hadoop-MapReduce model," International Journal of Reviews in Computing, vol. 12, 2012.

[25] X. Lin, "Mr-apriori: Association rules algorithm based on mapreduce," in 2014 IEEE 5th international conference on software engineering and service science, 2014.

[26] M. L. Massie, B. N. Chun, D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," Parallel Computing, vol. 30, no. 7, pp. 817--840, 2004.

[27] D. Kundu, S. I. Lavl, Cacti 0.8 network monitoring, Packt Publishing Ltd, 2009.

[28] S. B. Joshi, "Apache hadoop performance-tuning methodologies and best practices," in Proceedings of the 3rd acm/spec international conference on performance engineering, 2012.

[29] B. J. Mathiya, V. L. Desai, "Apache Hadoop Yarn Parameter configuration Challenges and Optimization," in 2015 International Conference on Soft-Computing and Networks Security (ICSNS), 2015.

[30] C. Li, H. Zhuang, K. Lu, M. Sun, J. Zhou, D. Dai, X. Zhou, "An adaptive auto-configuration tool for hadoop," in 19th International Conference on Engineering of Complex Computer Systems, 2014.

[31] H. Herodotou, Y. Chen, J. Lu, "A Survey on Automatic Parameter Tuning for Big Data Processing Systems," ACM Computing Surveys (CSUR), vol. 53, no. 2, pp. 1--37, 2020.

[32] D. Cheng, J. Rao, Y. Guo, X. Zhou, "Improving mapreduce performance in heterogeneous environments with adaptive task tuning," in Proceedings of the 15th International Middleware Conference, 2014.

[33] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A. R. Butt, N. Fuller, "Mronline: Mapreduce online performance tuning," in Proceedings of the 23rd international symposium on High-performance parallel and distributed computing, 2014.

[34] G. Liao, K. Datta, T. L. Willke, "Gunther: Search-based auto-tuning of mapreduce," in European Conference on Parallel Processing, 2013.

[35] S. Babu, "Towards automatic optimization of MapReduce programs," in Proceedings of the 1st ACM symposium on Cloud computing, 2010.

[36] T. T. Htay, S. Phyu, "Improving the performance of Hadoop MapReduce Applications via Optimization of concurrent containers per Node," in IEEE Conference on Computer Applications (ICCA), 2020.

[37] J. Liu, S. Tang, G. Xu, C. Ma, M. Lin, "A Novel Configuration Tuning Method Based on Feature Selection for Hadoop MapReduce," IEEE Access, vol. 8, pp. 63862--63871, 2020.

[38] H. Yang, Z. Luan, W. Li, D. Qian, G. Guan, "Statistics-based workload modeling for mapreduce," in IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2012.

[39] K. Wang, X. Lin, W. Tang, "Predator An experience guided configuration optimizer for Hadoop MapReduce," in 4Th IEEE international conference on cloud computing technology and science proceedings, 2012.

[40] N. Yigitbasi, T. L. Willke, G. Liao, D. Epema, "Towards machine learning-based auto-tuning of mapreduce," in IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, 2013.

[41] X. Ding, Y. Liu, D. Qian, "Jellyfish: Online performance tuning with adaptive configuration and elastic container in hadoop yarn," in IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), 2015.

[42] D. Wu, A. Gokhale, "A self-tuning system based on application profiling and performance analysis for optimizing hadoop mapreduce cluster configuration," in 20th Annual International Conference on High Performance Computing, 2013.

[43] Z. Bei, Z. Yu, H. Zhang, W. Xiong, C. Xu, L. Eeckhout, S. Feng, "RFHOC: a random-

Forest approach to auto-tuning Hadoop's configuration," IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 5, pp. 1470--1483, 2015.

[44] A. Gandomi, A. Movaghar, M. Reshadi, A. Khademzadeh, "Designing a MapReduce performance model in distributed heterogeneous platforms based on benchmarking approach," The Journal of Supercomputing, pp. 1--27, 2020.

[45] S. Kavulya, J. Tan, R. Gandhi, P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010.

[46] P. Lama, X. Zhou, "Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud," in Proceedings of the 9th international conference on Autonomic computing, 2012.

[47] C. Chen, Y. Zhuo, C. Yeh, C. Lin, S. Liao, "Machine learning-based configuration parameter tuning on hadoop system," in IEEE International Congress on Big Data, 2015.

[48] S. Ceesay, A. Barker, Y. Lin, "Benchmarking and Performance Modelling of MapReduce Communication Pattern," arXiv preprint arXiv:2005.11608, 2020.

[49] Hadoop Monitoring, https://github.com/kovyrin/hadoop-cacti-jtg/blob/master/src/com, [Accessed 11 12 2020].

Table 4: The Most Influential Parameters In Hadoop Systems.

#	Parameter Name	Type	Default value	Description	Examples of tuned values
1	dfs.replication	HDFS	3	Block replication	2,3,4
2	dfs.block.size	HDFS	64MB	HDFS block size	128MB, 256MB,374MB
3	mapreduce.task.io.sort.mb	Memory	100MB	The total amount of buffer memory to use while sorting files	100MB, 220MB, 240MB, 280MB
4	mapreduce.task.io.sort.factor	Memory	10	The number of streams to merge at once while sorting files	200, 300, 50, 80, 30, 10-100
5	mapreduce.map.sort.spill. percent	Memory	0.8	The soft limit in the serialization buffer	0.6, 0.67, 0.95
6	io.file.buffer.size	Memory	4KB	The size of the buffer for use in sequence files	8KB, 32KB
7	mapreduce.job.reduces	CPU	1	The default number of reduce tasks per job	16, 32-80, 99, 14, 1-1000
8	mapreduce.job .maps	CPU	2	The default number of map tasks per job	400, 676, 752
9	mapred.child.java.opts	Memory	200MB	Memory heap size childfern process	500MB, 800MB,1 GB
10	mapreduce.reduce.shuffle. parallelcopies	CPU	5	The default number of parallel transfers run by reduce during the copy(shuffle) phase	8, 10

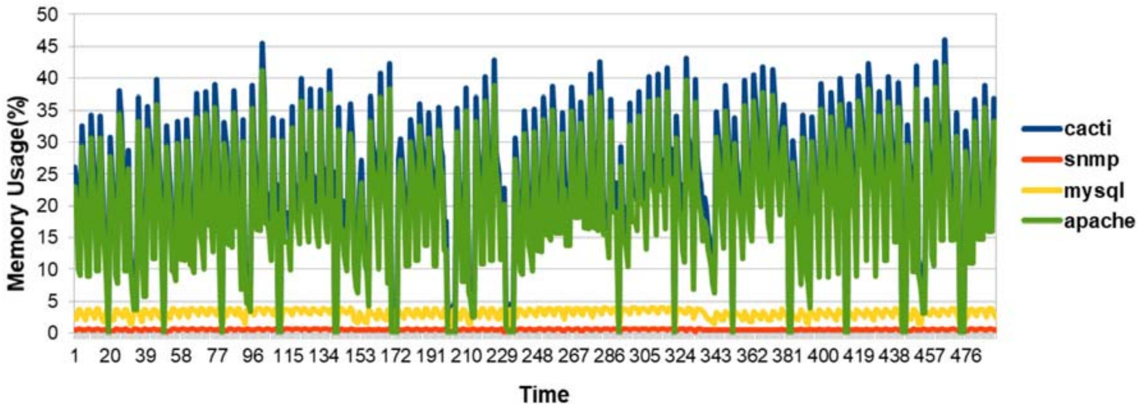


Figure 15: Memory Utilization. A High Memory Consumption By Cacti And Apache HTTP Server, Which Is Sometimes Going As High As A 45%.

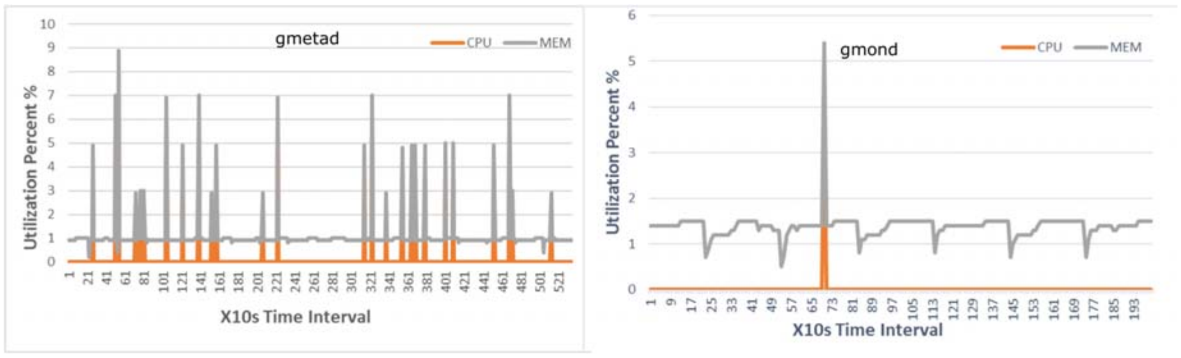


Figure 17: CPU And Memory Utilization Using Ganglia Gmetad And Gmond Daemons For 10 Seconds Time Period.

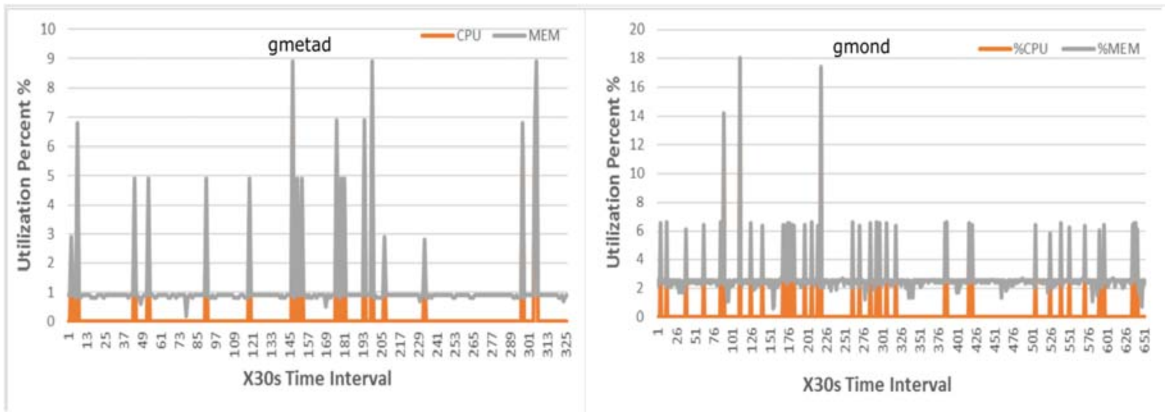


Figure 18: CPU And Memory Utilization Using Ganglia Gmetad And Gmond Daemons For 30 Seconds Time Period.

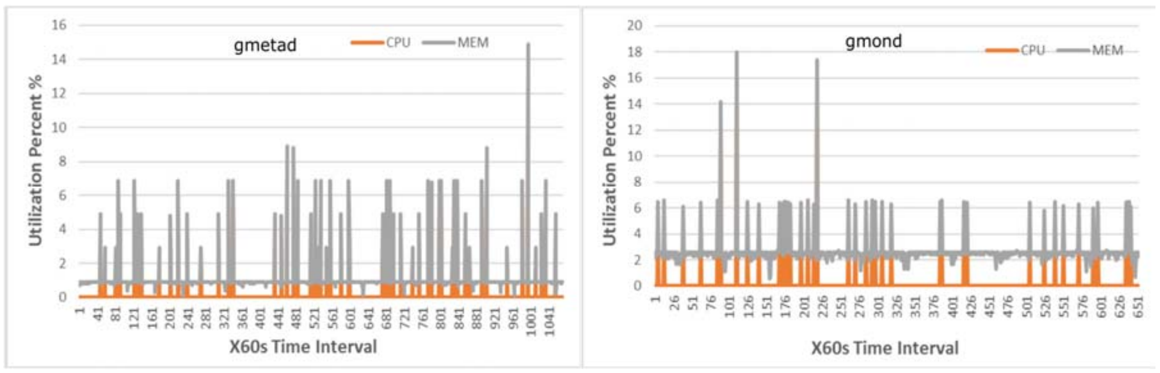


Figure 16: CPU And Memory Utilization Using Ganglia Gmetad And Gmond Daemons For 60 Seconds Time Period.