energies

MDPI

Article

Exploiting Coarse-Grained Parallelism Using Cloud Computing in Massive Power Flow Computation

Dong-Hee Yoon [1], Sang-Kyun Kang [2], Minseong Kim [3] and Youngsun Han [4,*]

1 Department of New and Renewable Energy, Kyungil University, Gyeongsan 38428, Korea; dhyoon@kiu.ac.kr
2 School of Electrical Engineering, Kyungil University, Gyeongsan 38428, Korea; sgkang@kiu.ac.kr
3 Department of Electrical and Computer Engineering, Korea University, Seoul 02841, Korea; minseong@korea.ac.kr
4 School of Electronic Engineering, Kyungil University, Gyeongsan 38428, Korea
* Correspondence: youngsun@kiu.ac.kr; Tel.: +82-53-600-5549

Received: 30 July 2018; Accepted: 27 August 2018; Published: 29 August 2018

Abstract: We present a novel architecture of parallel contingency analysis that accelerates massive power flow computation using cloud computing. It leverages cloud computing to investigate huge power systems of various and potential contingencies. Contingency analysis is undertaken to assess the impact of failure of power system components; thus, extensive contingency analysis is required to ensure that power systems operate safely and reliably. Since many calculations are required to analyze possible contingencies under various conditions, the computation time of contingency analysis increases tremendously if either the power system is large or cascading outage analysis is needed. We also introduce a task management optimization to minimize load imbalances between computing resources while reducing communication and synchronization overheads. Our experiment shows that the proposed architecture exhibits a performance improvement of up to 35.32× on 256 cores in the contingency analysis of a real power system, i.e., KEPCO2015 (the Korean power system), by using a cloud computing system. According to our analysis of the task execution behaviors, we confirmed that the performance can be enhanced further by employing additional computing resources.

Keywords: contingency analysis; power flow computation; coarse-grained parallelism; cloud computing; task management optimization

1. Introduction

As the complexity of power systems increases with the demand for electric power, the use of advanced power system equipment is becoming more widespread. Furthermore, power systems have become significantly more complicated following the integration of renewable energy resources, such as solar and wind [1]; the employment of high-speed operating devices, such as flexible AC transmission systems (FACTS) and high voltage direct current (HVDC) [2]; and the increasing prevalence of electric vehicles. In addition, although end users have only acted as electricity consumers in previous decades, smart grid environments enable them to take advantage of information technology (IT) and advanced power system facilities to become independent electric power suppliers [3]. In particular, the adoption of an energy storage system (ESS) provides opportunities for installing a variety of distributed power sources in the grid [4]. Hence, it is plausible that the complexity of power systems will increase continuously.

Contingency analysis, which is used to assess the effects of power system component failures in advance, is conducted to enable better power system management. Unfortunately, the number of computations required to calculate the power flow increases with the complexity of the power system [5]. There have been many studies on methods and applications of power flow computations,

and the Newton–Raphson (N-R) [6] and Gauss–Seidel (G-S) [7] methods have been adopted widely for this purpose. These computations take a long time even though the performance of computing devices is improving continuously. This is because one of the key steps of power flow computation is to solve nonlinear equations.

The computation time increases with the scale and complexity of a power system. In large power systems, many contingencies must be analyzed because the operation of the system is predicted for each possible contingency. Thus, the analysis is extremely time-consuming. Although $n-1$ contingencies are assumed for power system analysis in general, analyses of $n-k$ contingencies (several devices failing at the same time) and $n-1-1$ contingencies (a sequence of accidents) are also necessary for real power system operation [8]. In the case of $n-k$ contingency analysis, we can assume that k components fail at the same time. In this case, the large number of possible combinations of faults means that many calculations are required. In the case of $n-1-1$ contingencies, two devices fail sequentially. Therefore, the number of possible $n-1-1$ contingencies is equal to n multiplied by $n-1$.

Many calculations, requiring high performance computing, are necessary for power system analysis [9]. We can improve the speed of these calculations by applying parallel processing. Various methods for applying parallel processing for power flow computation have been studied previously. The basic concepts of using parallel processing for power flow computations are presented in [10, 11]. Following the publication of those papers, there have been some studies on parallel power flow computations in high speed computing environments [12–15]. In recent years, several studies have analyzed power systems by constructing an environment that physically uses a large number of cores [16,17].

The parallelism method of calculation, as an alternative to employing a number of cores, has also been studied extensively. Analysis of a power system requires many calculations related to the matrix and a number of studies have focused on parallel processing of matrix calculations [18,19]. In addition, graphics processing units (GPUs) have been used for fast power flow calculations, as an alternative to other machine cores for parallel processing [20–23]. Detailed explanations of the parallel processing issues affecting early research are provided in Section 3.

Previous studies have already established that it is better to use parallel processing for analysis of power systems with high computational complexity. For efficient parallel processing, it is necessary to secure resources for calculation; however, physical expansion is limited by cost, and, even if a dedicated system is provided, there is the further disadvantage that the machine is idle when the analysis is not in progress. To overcome these drawbacks, we propose a parallel processing scheme for power system analysis using cloud computing, in which the necessary computing resources can be acquired at the required time. In recent years, technology has advanced to the stage where it is relatively easy to create an environment that enables researchers to take advantage of cloud computing.

Parallel processing can be divided into fine-grained parallelism for allocating parts of the operation and coarse-grained parallelism for parallel processing of the entire operation. Most previous studies on matrix handling have investigated fine-grained parallelism. In this study, we propose a cloud computing-based contingency analysis method that exploits coarse-grained parallelism so that the entire computation is parallel. We obtained a significant performance improvement in the contingency analysis of KEPCO2015 by using a Hadoop-based Amazon EC2 cloud computing system with 1 NameNode and 16 DataNodes.

The experimental results show that our proposed contingency analysis achieved speedups of 8.00×, 17.98×, 27.28×, and 35.32× on 32, 64, 128, and 256 cores, respectively, when adopting task management optimization. We also confirmed that the addition of computing resources can lead to further performance enhancement, even for more contingencies, because the total execution time decreases in proportion to the number of cores used.

In summary, the main contributions of this paper are:

- We propose a massive parallel contingency analysis using a cloud computing platform.

- We propose a task management optimization to improve performance and scalability of the parallel contingency analysis on a Hadoop-based cloud computing system.
- We obtained a 37-fold or greater performance improvement on the Amazon Web Service (AWS) environment.

The rest of the paper is organized as follows: In Section 2, we describe the background of our study and briefly explain cloud computing, parallelism granularity, and power flow computation. We present related works in Section 3. In Section 4, we introduce the overall architecture of our contingency analysis method, the N-R method, and the motivation for this study. In Section 5, we provide a detailed description of a massive cloud computing-based power flow computation. We evaluate the performance of our method in terms of speedup and present task execution behavior in Section 6. Finally, conclusions are provided in Section 7.

2. Background

2.1. Cloud Computing

Cloud computing is an Internet-based computing technology that facilitates ubiquitous access to a shared and distributed pool of configurable computing resources such as storage, networks, servers, applications, services, and so on [24–27]. Cloud computing allows users to process and store data in a privately owned cloud or third-party server located within a data center. It improves the performance of applications through enhanced manageability and the requirement for less maintenance; thus, it also contributes to meeting additional user requirements.

Figure 1 presents a brief conceptual diagram of cloud computing. The cloud computing environment primarily consists of infrastructure, platform, and software; on-demand computing resources are delivered to clients in the form of these services such as Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [28,29].

- SaaS: Cloud-based applications that are run on remote computers in the cloud and are maintained by a third party. In general, users access these applications via their web browsers.
- PaaS: All of the resources are provided to deliver web-based cloud applications without the cost and complexity associated with establishing and maintaining the underlying hardware, software, provisioning, and hosting.
- IaaS: Computing resources of infrastructure, including storage, network, and servers, are provided to clients on a pay-per-use basis.

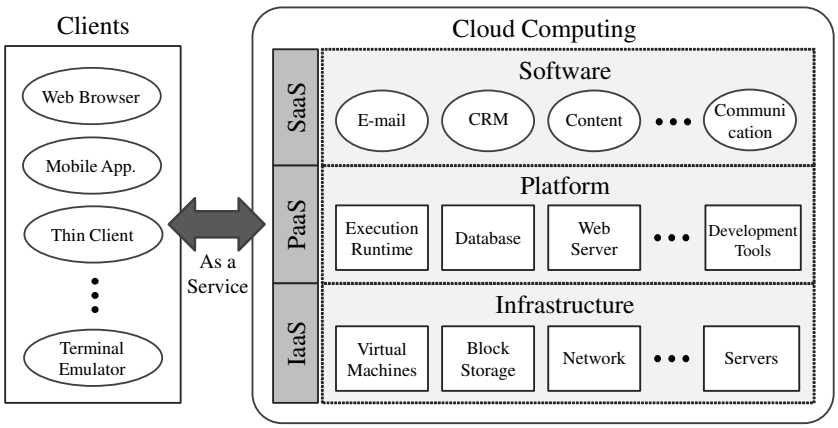


Figure 1. Conceptual diagram of cloud computing.

2.2. Parallelism Granularity

In parallel computing, parallelism granularity indicates a qualitative measure of the ratio of computation to communication [30]. Synchronization events usually separate periods of computation from periods of communication. The granularity of parallelism is mainly classified into two categories based on the amount of work that is performed by a parallel task: fine-grained and coarse-grained [31,32].

2.2.1. Fine-Grained Parallelism

Figure 2 presents an example of obtaining fine-and coarse-grained parallelism from a single sequential program. As shown in Figure 2c, in fine-grained parallelism, a subprogram is divided into a large number of small tasks that are individually assigned to, and then executed by, multiple processors. As each of the tasks is small, and the tasks are distributed evenly across the processors, fine-grained parallelism significantly improves load balancing, but it also has the disadvantage of increasing communication and synchronization overheads. Hence, fine-grained parallelism is suitable for use with shared memory architectures, e.g., GPUs, that provide fast communication between multiple processors [33,34]. Programmers rarely make explicit use of this parallelism, so in general the parallelism is controlled by compiler technology [35].

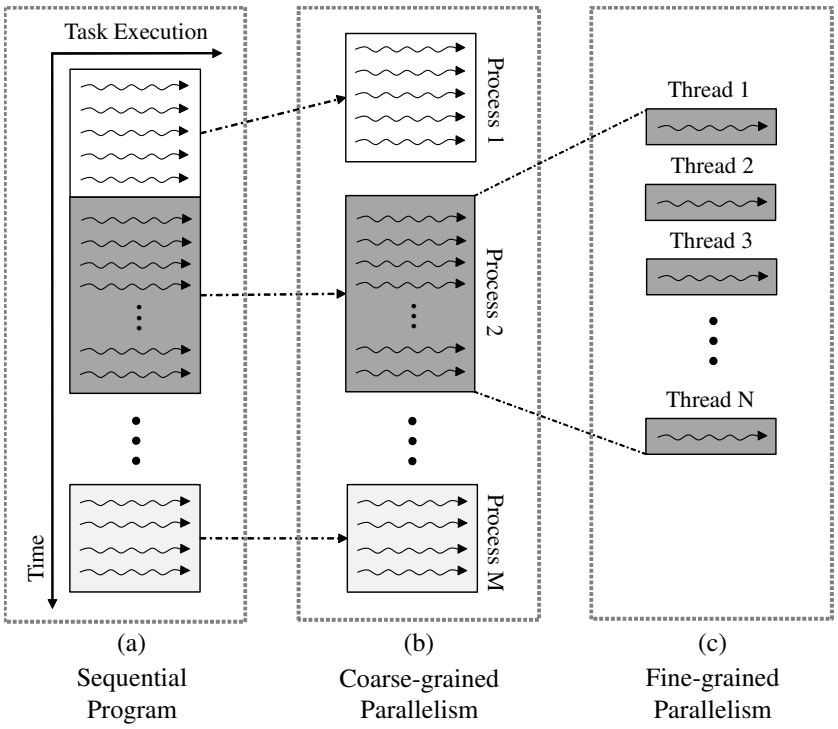


Figure 2. Example of fine- and coarse-grained parallelism. Arrows indicate task execution. In a sequential program, the tasks are executed from top to bottom in order over time.

2.2.2. Coarse-Grained Parallelism

Figure 2b illustrates that coarse-grained parallelism divides one program into large tasks. Since subprograms with a large number of tasks are assigned to different processors, the parallelism confers the benefits of low communication and synchronization overheads, but incurs load imbalance problems. Message-passing architecture is definitely suitable for coarse-grained parallelism, since it requires a long time to transfer data between multiple processes [36]. We need to separate and

parallelize a program ourselves based on a full understanding of its code; thus, it is somewhat challenging to exploit this parallelism [37].

3. Related Works

Typically, power system analysis is based on contingency analysis, which is in turn based on the results of power flow computations. As mentioned earlier, the time required to compute power flows increases with the size and complexity of the power system. In addition, the number of contingencies requiring analysis also tends to increase for stable power system operation.

As the performance of computers continues to improve, parallel processing is not necessary when conducting power flow computations for a small number of contingencies. However, when analyzing a large number of contingencies, it is advantageous to adopt parallel processing and thus reduce the computation time. For instance, it would be useful to perform $N - k$ contingency analysis in parallel. The computational complexity of this analysis is very large, so it is appropriate to process the power flow computations in parallel.

The basic concepts related to the application of parallel processing to power flow computation are explained in [10,11]. Parallel contingency analysis in high performance computing environments is studied in [12], where an effective method for conducting contingency analysis using parallel processing is presented. In [13], a dynamic computational load balancing method is proposed using high performance computing machines for a large number of computations.

There have been attempts to improve the performance of parallel computing using multiple machines. Recently, several large-scale projects using parallel computing have been performed for power system analysis. In [16], contingency analysis is performed using a commercial program in a massively parallel environment. A total of 4127 contingencies were scaled to 4127 cores and the solution time is reduced significantly. In [17], a computational platform for dynamic security assessment of a large power system with a commercial analysis tool is presented as a part of the iTesla project; 10,000 cores are used in the case study in this project.

The need for high computing power to analyze a power system and its effects is demonstrated by the results of previous studies. However, there is a physical constraint whereby many large-scale connected computing machines are needed. In this paper, we propose a power system analysis platform using cloud computing as a solution to this limit, without the need for a commercial analysis tool, and we obtain reasonable results without physical constraints by using the cloud computing environment.

There have also been a number of previous studies on how to distribute calculations for parallel processing, and most previous research has focused on applying matrix calculation to parallel computing [10,18,19]. These studies are based on fine-grained parallelism that assigns some of the calculations. Another method for exploiting fine-grained parallelism is to perform power flow computations using GPUs. The GPU-based technique is an example of a parallel processing method that increases the speed of power flow computations. In [20], the basics of parallel computing using GPUs are established, and a method for utilizing GPUs is presented in [21]. A detailed implementation of power flow computation parallel processing done using GPUs is provided in [22,23], with reasonable results. Using GPUs is an effective method for reducing the time taken for power flow computations, but does not exploit the coarse-grained method of assigning all calculations. In addition, there is a limit to the increase in the speed because the overall effect of the parallel processing depends on the performance of the hardware. To overcome these weaknesses, we herein propose a method to perform power flow computations in a cloud computing environment without specific hardware configurations and exploit coarse-grained parallelism to perform entire computations in parallel.

4. Challenges in Contingency Analysis of Real Power Systems

4.1. Overall Architecture of Contingency Analysis

Algorithm 1 describes the structure of a conventional contingency analysis that performs the power flow computation of a power system for N different contingencies. The inputs of the algorithm are Ctg and D^{in}, which denote a list of contingency data and cluster data of the power system, respectively. The output is D^{out}, which is a list of convergence results of the power system for all contingencies. D^{tmp} indicates the result of applying one contingency to the cluster data and D_i^{out} is the result of the i-th power flow computation.

This analysis algorithm mainly consists of the following two procedures. First, it updates the cluster data of a power system using the i-th contingency data, i.e., Ctg_i, at Line 5. Second, it obtains a convergence result of the updated cluster data by performing a power flow computation at Line 7. Since the algorithm repeats the previous two procedures for all contingency data, its execution time linearly increases in proportion to the number of contingencies, i.e., N.

Algorithm 1 Contingency analysis.

INPUT Ctg: list of contingency data, D^{in}: cluster data.
OUTPUT D^{out}: list of convergence results.

```
1: procedure ContingencyAnalysis(Ctg, D^in)
2:     i = 0;
3:     while i < N do
4:         // Updates cluster data with i-th contingency
5:         D^tmp = UpdateClusterData(D^in, Ctg_i);
6:         // Performs i-th power flow computation
7:         D_i^out = PowerFlowComputation(D^tmp);
8:     end while
9: end procedure
```

4.2. Newton–Raphson Based Power Flow Computation

The N-R method can be used to determine the magnitude and angle of the voltage at each bus in the power system. A brief description of the N-R method and its application is as follows.

The general form of the nodal equation for a power system network is:

$$I = Y_{bus}V \tag{1}$$

where Y_{bus} is the admittance matrix of the network, I is a current vector, and V is a voltage vector. The ith equation, describing the flow at bus i, is:

$$I_i = \sum_{k=1}^{n} Y_{ik}Vk \tag{2}$$

A complex power equation can be expressed as follows:

$$S_i = V_i(\sum_{k=1}^{n} Y_{ik}Vk)^* = V_i \sum_{k=1}^{n} Y_{ik}^*Vk^* = P_i + jQ_i \tag{3}$$

It is necessary to derive power balance equations before we can solve the power flow problem. The method for deriving power balance equations is discussed in detail in [38], so we only provide the derived equations. At bus i, the power balance equations are

$$P_i = \sum_{k=1}^{n} |V_i||V_k|(G_{ik}\cos\theta_{ik} + B_{ik}\sin\theta_{ik}) \tag{4}$$

$$Q_i = \sum_{k=1}^{n} |V_i||V_k|(G_{ik}\sin\theta_{ik} - B_{ik}\cos\theta_{ik}) \tag{5}$$

The N-R method is an iterative technique for solving a set of nonlinear equations. We can use Equations (4) and (5), which are nonlinear power balance equations, to calculate the power flow solutions. The N-R method is described in detail in [38]. The matrix form of the resulting linear set of equations is:

$$\Delta f = J\Delta x \tag{6}$$

where J is the Jacobian. We can obtain the complete formulation of the power mismatch using the power balance equations in Equations (4) and (5) and the N-R method from Equation (6), as follows:

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial P}{\partial \theta} & \frac{\partial P}{\partial V} \\ \frac{\partial Q}{\partial \theta} & \frac{\partial Q}{\partial V} \end{bmatrix}}_{\text{Jacobian}} \begin{bmatrix} \Delta \theta \\ \Delta V \end{bmatrix} \tag{7}$$

The N-R method converges to the solution after fewer iterations than the G-S method, and is more accurate. In addition, it exhibits better convergence during the computation, and is more efficient in large-scale systems. However, it usually requires more computation per iteration than other power flow computation methods, so the computation time per iteration is slightly longer. To improve the performance and feasible size of the computations by implementing parallel processing, we selected the N-R method for the power flow computations in this study.

4.3. Motivation

Figure 3 shows the execution time of contingency analysis for a variety of power systems, including a real power system, i.e., KEPCO2015. For each power system, we performed the contingency analysis by applying the contingencies from each branch in turn. In other words, we computed the power flow of a power system with $N-1$ branches, where N is the total number of branches of the system. Hence, we employed 17, 32, 64, 170, 306, and 2589 contingencies to evaluate the execution time of contingency analysis for power systems, IEEE14, IEEE24, IEEE57, IEEE118, IEEE300, and KEPCO2015, respectively. The specifications of the power systems are briefly outlined in Table 1, which describes how many buses, generators, and branches are involved in each system.

These results show that the contingency analysis takes less than 1 s in most of the IEEE systems, except for IEEE300, which requires up to 10.914 s. However, we obtained a completely different result for KEPCO2015, which represents the state of a real power system located in South Korea in 2015. Due to its large-scale cluster data including buses, branches, generators, and so on, the contingency analysis requires a considerably long execution time, i.e., 1636.589 s. Further, if we assume that contingencies from more than two different branches can take place at the same time, the execution time increases significantly because the number of contingencies to be analyzed grows exponentially. The contingency analysis in most real power systems consumes a significantly large amount of execution time, and acceleration of this analysis should be studied.

In this paper, we propose a parallelized contingency analysis using cloud computing by exploiting the coarse-grained parallelism of independent contingencies from each other.

Table 1. Brief Specification of Power Systems.

Power System	Buses	Generators	Branches
IEEE14	14	5	17
IEEE24	24	11	32
IEEE57	57	7	64
IEEE118	118	54	170
IEEE300	300	69	306
KEPCO2015	1946	390	2589

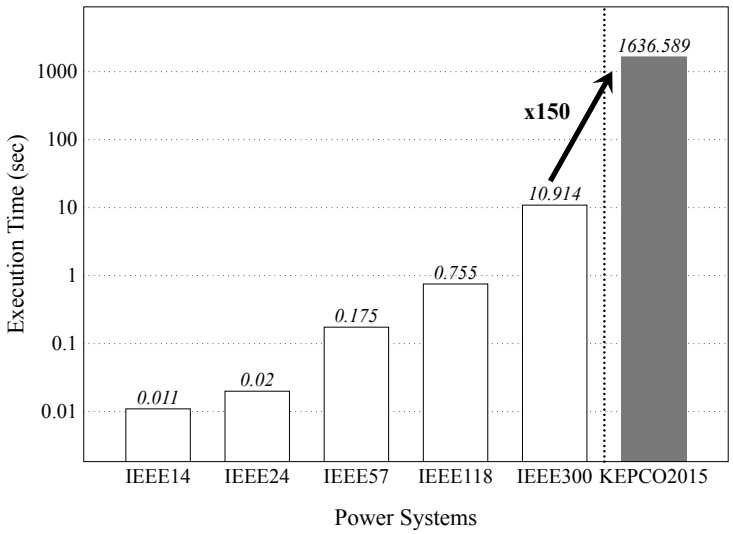


Figure 3. Execution time of conventional power system contingency analysis. For each power system, we obtained the results by evaluating the effects of the contingencies from each branch in turn.

5. Massive Power Flow Computation Using Cloud Computing

In this section, we describe how to perform massive power flow computations using cloud computing. We explain how to obtain coarse-grained parallelism of the power flow computation with the MapReduce framework. In addition, we present a case study that performs parallel contingency analysis on a Hadoop platform [39].

5.1. Exploiting Coarse-Grained Parallelism with MapReduce

Some previous works on power flow analysis parallelization have obtained fine-grained parallelism by parallelizing a conjugate gradient algorithm for solving a linear system, i.e., a system of the form $Ax = b$, which is the most time-consuming part of the power flow analysis. However, performance improvements derived from the fine-grained parallelism are restricted by computing resource limitations if a large power flow analysis, e.g., contingency analysis, is performed. In addition, some other studies have presented to exploit coarse-grained parallelism by independently performing multiple contingency analysis on multi-processors in parallel. Although they employ well-known parallel programming models such as OpenMP (Open Multi-Processing), MPI (Message Passing Interface), and so on, it is still very challenging for users to obtain significant coarse-grained parallelism due to inevitable load imbalance and communication overhead.

We propose to resolve this problem by exploiting coarse-grained parallelism within a MapReduce framework on cloud computing. We can significantly improve the performance of a massive power flow analysis by simultaneously executing multiple independent tasks via the enormous computing capability of cloud computing. In particular, since most of the recent MapReduce-based cloud computing environments also provide GPU-based computing capabilities that make it easy to

implement fine-grained parallelism [40], we were able to maximize the performance of our method by applying it orthogonally to the existing fine-grained parallelism. MapReduce is one of the most prominent programming models used to produce and manipulate large datasets by executing parallel and distributed algorithms on computing clusters. In general, a MapReduce system is designed to be maintained on many distributed servers, to perform multiple tasks in parallel, to manage communication between the different components of the system, and to support redundancy and fault tolerance.

Figure 4 presents a brief overview of a MapReduce program. A MapReduce program principally consists of a map function that filters and sorts the input data, and a reduce function that integrates the intermediate data from the map function into the output data. Each split input item is assigned to a mapper by a task scheduler, and the mapping results are output to intermediate files. The files are shuffled, partitioned, and then input to the reducers, which assemble the output data. To obtain significant coarse-grained parallelism, we need to divide the input data to minimize communication and synchronization overheads so that each of the mappers performs a large number of tasks independent from each other.

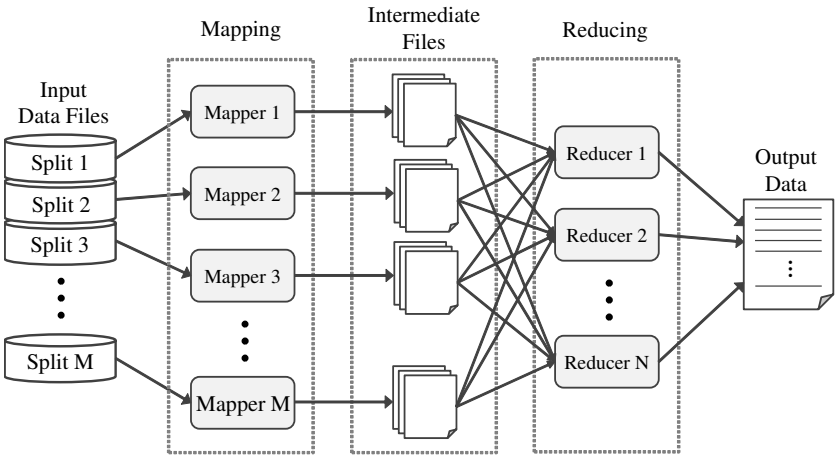


Figure 4. Overall architecture of a MapReduce program.

5.2. Case Study: Hadoop-Based Parallel Contingency Analysis

Our parallel contingency analysis program, which uses Hadoop infrastructure to perform massive power flow computations, is outlined schematically in Figure 5. The input data is stored in a Hadoop distributed file system (HDFS) comprised of the following two types of data: a set of contingency data and a set of cluster data. Each contingency is assigned to a different mapper, but the cluster data are shared by all of the mappers.

First, each mapper updates the cluster data with the given contingency data, and then performs power flow analysis to estimate the convergence of the power system specified by the cluster data. The mapper returns a key and value pair, i.e., a contingency ID (CID) and the maximum mismatch, for each iteration of the N-R algorithm. In our implementation, the N-R algorithm is implemented in C/C++ and is accessed using a Java native interface (JNI) in the mapper, i.e.,the map method. Second, the intermediate results from the mappers are delivered to reducers after being shuffled and partitioned. Then, each reducer takes a pair of key and list values, i.e., a CID and a list of mismatch values, and estimates whether the cluster converges at the contingency that is indicated by the CID. The reducer also returns a key and value pair, i.e., a CID and the result of the convergence analysis result. Finally, the results of the convergence analysis for all contingencies are merged together and written to an output file.

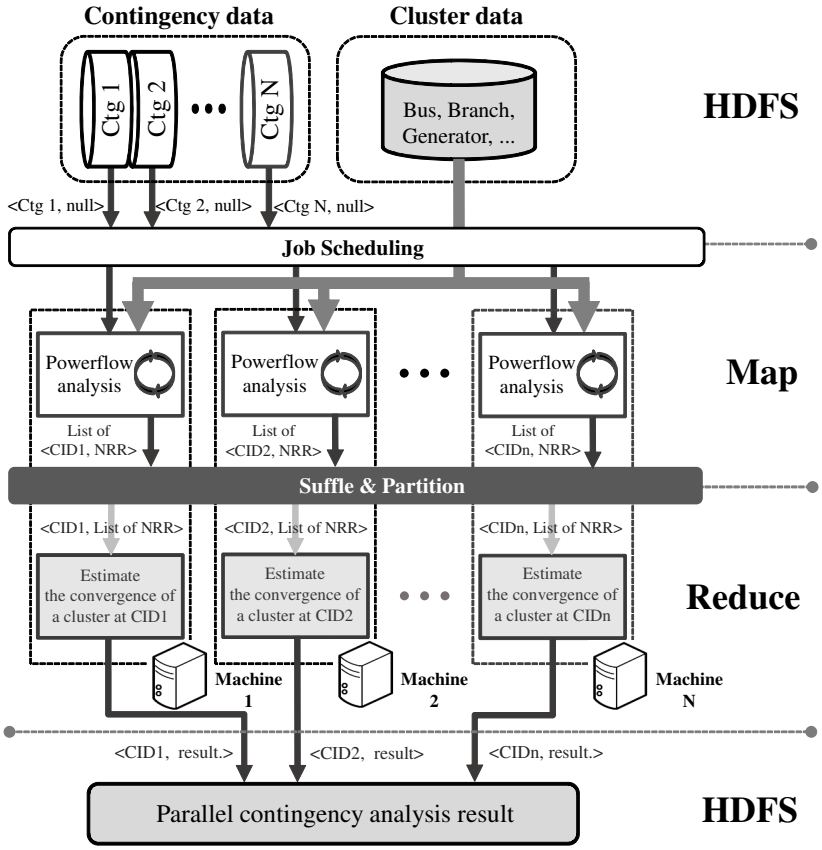


Figure 5. Hadoop-based parallel contingency analysis. *Ctg* denotes a contingency scenario and an NRR is a result derived from an iteration of the Newton–Raphson (N-R) algorithm.

6. Performance Evaluation

In this section, we describe the experimental environment in detail and evaluate the performance of our Hadoop-based parallel contingency analysis.

6.1. Experimental Environment

In Table 2, we describe the organization of the Hadoop-based cloud computing system that we employed to assess the performance of our proposed parallel contingency analysis. We assembled the Hadoop system using remote computing resources from Amazon Elastic Compute Cloud (EC2), which are provided by AWS. We used Cloudera's open-source Apache Hadoop distribution, i.e., CDH, which provides an Apache Hadoop framework such as Hadoop YARN, HDFS, and so on.

In detail, the Hadoop system occupies a single NameNode and 16 DataNodes. A NameNode is designed to provide a unified namespace that the HDFS uses to manage access to the data files in the distributed file system. It also supports file system operations, such as open, close, and rename, and maps split data files, i.e., data blocks, to DataNodes. The DataNodes redundantly store the data blocks in their storage, execute read and write operations on the blocks, and implement the map and reduce procedures.

For the evaluation, we designed each of the DataNodes to utilize 2, 4, 8, and 16 cores of an Intel Xeon E5-2686 v4. Hence, we performed the experiments with a large number of cores, i.e., 32, 64, 128, and 256, by using 16 DataNodes together. Further, since we adopted *M*4 instances of AWS EC2, we expected to use different network bandwidths of 450, 750, 1000, and 2000 Mbps when using 32, 64, 128, and 256 cores, respectively. We performed 16 reduce tasks, which we started after more than 90% of all map tasks were complete.

Table 2. Organization of our Hadoop-based cloud computing system.

Feature	Description
Cloud computing platform	Amazon EC2
Hadoop platform	Cloudera Manager, CDH 5.2
Hadoop framework	Hadoop YARN MR2, HDFS
Hadoop NameNode	1 NameNode with 4 cores
Hadoop DataNode	16 DataNodes with 2, 4, 8, 16 cores each
Network bandwidth	450, 750, 1000, 2000 Mbps

6.2. Task Management Optimization

As mentioned in Section 2.2, to maximize the performance improvement from coarse-grained parallelism in parallel contingency analysis, we are required to minimize load imbalance among computing resources while diminishing the overheads of communication and synchronization [41]. We propose a task management optimization that splits all contingencies into as many chunks as there are available cores in advance, and then assigns each of the chunks to a map task to be executed on a single core. By adopting optimization instead of employing a single map task for each contingency, we can improve the performance degradation caused by task management overhead on a Hadoop platform.

6.3. Experimental Results

Figure 6 shows the performance improvement of our parallel contingency analysis of KEPCO2015 on 32, 64, 128, and 256 cores with respect to conventional contingency analysis. The gray bar represents the speedup when adopting task management optimization and the white bar shows the results with no optimization. Without the optimization, we obtained speedups of 1.78×, 3.26×, 6.62×, and 12.85× on 32, 64, 128, and 256 cores, respectively. The speedup improved by slightly more than a factor of two because the network bandwidth increased from 750 to 1000 Mbps when the number of cores increased from 64 to 128.

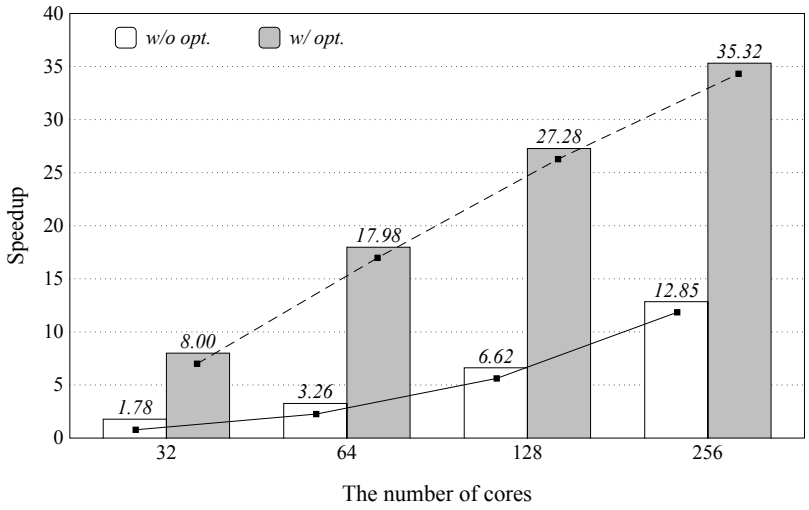
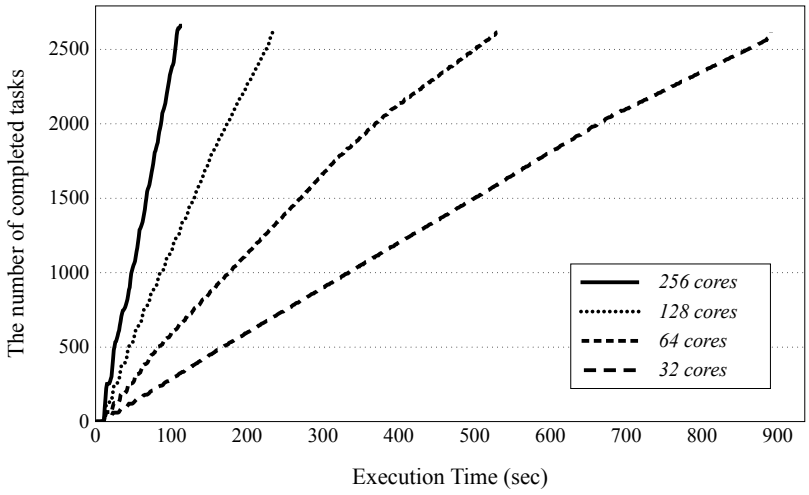


Figure 6. Performance improvement of Hadoop-based contingency analysis for KEPCO2015 while increasing the number of cores from 32 to 256. The gray and white bars indicate, respectively, the speedup obtained with and without application of the task management optimization described in Section 6.2.

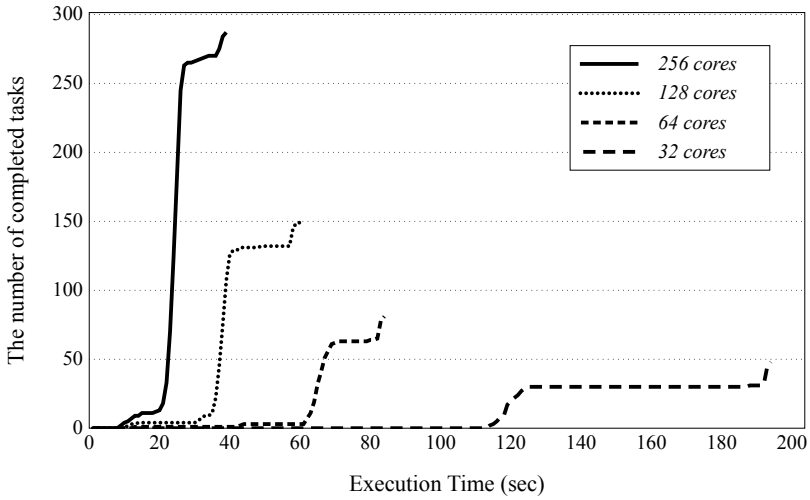
Furthermore, the task management optimization strategy significantly enhanced the performance of the parallel contingency analysis. We improved the performance by up to 35.32× on 256 cores and

achieved the speedup of 8.00×, 17.98×, and 27.28× on 32, 64, and 128 cores, respectively. These results confirm that our proposed architecture yields significant performance improvements by exploiting the coarse-grained parallelism of contingency analysis using cloud computing. In particular, as the speedup increases proportionally to the number of cores used, we expect that additional computing resources will lead to further performance enhancements.

Figure 7 shows the task execution behavior of our parallel contingency analysis of KEPCO2015, both with and without applying the task management optimization. Each line indicates the total completed tasks over time for the different numbers of cores, i.e., 32, 64, 128, and 256 cores. The completed tasks involve both map and reduce tasks. As shown in Figure 7a, when the optimization was not adopted, the number of completed tasks became identical to the number of contingencies in KEPCO2015, i.e., 2589, and the total execution time decreased in proportion to the number of employed cores, which increased from 32 to 256.



(**a**) Without optimization



(**b**) With optimization

Figure 7. The accumulated number of tasks completed over time during the parallel contingency analysis of KEPCO2015: (**a**) without; and (**b**) with task management optimization.

Figure 7b shows that the number of completed tasks becomes equal to the number of occupied cores through task management optimization. This is because we created and executed as many map tasks as the number of cores, as mentioned in Section 6.2. We can also confirm that the optimization

reduced the overall execution time for completing all tasks significantly. When using 256 cores, the number of completed tasks increased relatively sharply in the first round, i.e., between 19 s and 29 s, and then rose gradually over the next few seconds of the second round. In detail, after 90% of all the map tasks were completed in the first round, 16 reduce tasks were created and executed in the second round. Finally, from 33 s onwards, the number of completed tasks started to rise quickly again as the reduce tasks began to finish. Although different numbers of cores were used, similar behaviors are shown in each graph.

7. Conclusions

In this paper, we propose a novel parallel architecture for accelerating contingency analysis in real power systems. To the best of our knowledge, our study is the first to suggest a cloud computing platform for massive power flow computations by exploiting coarse-grained parallelism. We also propose a task management optimization that divides all contingencies into as many chunks as there are available cores in advance, and then assigns each of the chunks to a different core. This optimization allows minimizing load imbalances between computing resources while diminishing the overheads imposed by communication and synchronization.

As a result, we obtained a significant performance improvement in the contingency analysis of a real power system, i.e., KEPCO2015, by using a Hadoop-based cloud computing system with 1 NameNode and 16 DataNodes on Amazon EC2. The experimental results show that our proposed contingency analysis achieved speedups of 8.00×, 17.98×, 27.28×, and 35.32× on 32, 64, 128, and 256 cores, respectively, when adopting task management optimization. Furthermore, from task execution behaviors, we can expect to enhance the performance further by employing additional computing resources, since the total execution time of tasks is decreased in proportion to the number of cores used.

Different from the conventional fine- and coarse-grained parallelization technologies with ultimately limited hardware resources such as GPU, local servers, and so on, our proposed architecture can tremendously accelerate the analysis of large power systems that require massive power flow computations by using unlimited cloud computing.

Author Contributions: All authors contributed to this work. D.-H.Y.: Writing—Original Draft Preparation; M.K.: Performance Evaluation; and S.-K.K. and Y.H.: Supervision and Guidance.

Funding: This research was funded by Korea Electric Power Corporation (grant number: R17XA05-48).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alizadeh, M.; Moghaddam, M.P.; Amjady, N.; Siano, P.; Sheikh-El-Eslami, M. Flexibility in future power systems with high renewable penetration: A review. *Renew. Sustain. Energy Rev.* **2016**, *57*, 1186–1193. [CrossRef]
2. Debnath, S.; Qin, J.; Bahrani, B.; Saeedifard, M.; Barbosa, P. Operation, control, and applications of the modular multilevel converter: A review. *IEEE Trans. Power Electron.* **2015**, *30*, 37–53. [CrossRef]
3. Tuballa, M.L.; Abundo, M.L. A review of the development of Smart Grid technologies. *Renew. Sustain. Energy Rev.* **2016**, *59*, 710–725. [CrossRef]
4. Luo, X.; Wang, J.; Dooner, M.; Clarke, J. Overview of current development in electrical energy storage technologies and the application potential in power system operation. *Appl. Energy* **2015**, *137*, 511–536. [CrossRef]
5. Chen, Y.; Huang, Z.; Liu, Y.; Rice, M.J.; Jin, S. Computational Challenges for Power System Operation. In Proceedings of the 2012 45th Hawaii International Conference on System Sciences, Maui, HI, USA, 4–7 January 2012; pp. 2141–2150. [CrossRef]
6. Mumtaz, F.; Syed, M.; Al Hosani, M.; Zeineldin, H. A novel approach to solve power flow for islanded microgrids using modified newton raphson with droop control of dg. *IEEE Trans. Sustain. Energy* **2016**, *7*, 493–503. [CrossRef]

7. Stott, B. Review of load-flow calculation methods. *Proc. IEEE* **1974**, *62*, 916–929. [CrossRef]
8. Mitra, P.; Vittal, V.; Keel, B.; Mistry, J. A Systematic Approach to *n*-1-1 Analysis for Power System Security Assessment. *IEEE Power Energy Technol. Syst. J.* **2016**, *3*, 71–80. [CrossRef]
9. Falcao, D.M.; Borges, C.L.; Taranto, G.N. High performance computing in electrical energy systems applications. In *High Performance Computing in Power and Energy Systems*; Springer: Berlin, Germany, 2013; pp. 1–42.
10. Wu, J.Q.; Bose, A. Parallel solution of large sparse matrix equations and parallel power flow. *IEEE Trans. Power Syst.* **1995**, *10*, 1343–1349.
11. Fukuyama, Y.; Nakanishi, Y.; Chiang, H.D. Parallel power flow calculation in electric distribution networks. In Proceedings of the 1996 IEEE International Symposium on Circuits and Systems (ISCAS 96), Atlanta, GA, USA, 15 May 1996; Volume 1, pp. 669–672.
12. Ezhilarasi, G.A.; Swarup, K.S. Parallel contingency analysis in a high performance computing environment. In Proceedings of the 2009 International Conference on Power Systems, Kharagpur, India, 27–29 December 2009; pp. 1–6. [CrossRef]
13. Huang, Z.; Chen, Y.; Nieplocha, J. Massive contingency analysis with high performance computing. In Proceedings of the Power & Energy Society General Meeting, Calgary, AB, Canada, 26–30 July 2009; pp. 1–8.
14. Jin, S.; Chen, Y.; Diao, R.; Huang, Z.H.; Perkins, W.; Palmer, B. Power grid simulation applications developed using the GridPACK^{TM} high performance computing framework. *Electr. Power Syst. Res.* **2016**, *141*, 22–30. [CrossRef]
15. Guerra, G.; Martinez-Velasco, J.A. Evaluation of MATPOWER and OpenDSS load flow calculations in power systems using parallel computing. *J. Eng.* **2017**, *2017*, 195–204. [CrossRef]
16. Smith, S.; Van Zandt, D.; Thomas, B.; Mahmood, S.; Woodward, C. *HPC4Energy Final Report: GE Energy*; Technical Report; Lawrence Livermore National Laboratory (LLNL): Livermore, CA, USA, 2014.
17. Konstantelos, I.; Jamgotchian, G.; Tindemans, S.H.; Duchesne, P.; Cole, S.; Merckx, C.; Strbac, G.; Panciatici, P. Implementation of a massively parallel dynamic security assessment platform for large-scale grids. *IEEE Trans. Smart Grid* **2017**, *8*, 1417–1426. [CrossRef]
18. Chan, K. Parallel algorithms for direct solution of large sparse power system matrix equations. *IEE Proc. Gener. Transm. Distrib.* **2001**, *148*, 615–622. [CrossRef]
19. Lehoucq, R.B.; Salinger, A.G. Large-scale eigenvalue calculations for stability analysis of steady flows on massively parallel computers. *Int. J. Numer. Methods Fluids* **2001**, *36*, 309–327. [CrossRef]
20. Garcia, N. Parallel power flow solutions using a biconjugate gradient algorithm and a Newton method: A GPU-based approach. In Proceedings of the Power and Energy Society General Meeting, Providence, RI, USA, 25–29 July 2010; pp. 1–4.
21. Li, X.; Li, F. GPU-based power flow analysis with Chebyshev preconditioner and conjugate gradient method. *Electr. Power Syst. Res.* **2014**, *116*, 87–93. [CrossRef]
22. Roberge, V.; Tarbouchi, M.; Okou, F. Parallel Power Flow on Graphics Processing Units for Concurrent Evaluation of Many Networks. *IEEE Trans. Smart Grid* **2017**, *8*, 1639–1648. [CrossRef]
23. Li, X.; Li, F.; Yuan, H.; Cui, H.; Hu, Q. GPU-based fast decoupled power flow with preconditioned iterative solver and inexact newton method. *IEEE Trans. Power Syst.* **2017**, *32*, 2695–2703. [CrossRef]
24. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; et al. A View of Cloud Computing. *Commun. ACM* **2010**, *53*, 50–58. [CrossRef]
25. Sadiku, M.N.O.; Musa, S.M.; Momoh, O.D. Cloud Computing: Opportunities and Challenges. *IEEE Potentials* **2014**, *33*, 34–36. [CrossRef]
26. Sadooghi, I.; Martin, J.H.; Li, T.; Brandstatter, K.; Maheshwari, K.; de Lacerda Ruivo, T.P.P.; Garzoglio, G.; Timm, S.; Zhao, Y.; Raicu, I. Understanding the performance and potential of cloud computing for scientific applications. *IEEE Trans. Cloud Comput.* **2017**, *5*, 358–371. [CrossRef]
27. Tchana, A.; De Palma, N.; Safieddine, I.; Hagimont, D. Software consolidation as an efficient energy and cost saving solution. *Future Gener. Comput. Syst.* **2016**, *58*, 1–12. [CrossRef]
28. Vaquero, L.M.; Rodero-Merino, L.; Caceres, J.; Lindner, M. A Break in the Clouds: Towards a Cloud Definition. *SIGCOMM Comput. Commun. Rev.* **2008**, *39*, 50–55. [CrossRef]
29. Teabe, B.; Tchana, A.; Hagimont, D. Enforcing CPU allocation in a heterogeneous IaaS. *Future Gener. Comput. Syst.* **2015**, *53*, 1–12. [CrossRef]

30. Chen, D.K.; Su, H.M.; Yew, P.C. The Impact of Synchronization and Granularity on Parallel Systems. *SIGARCH Comput. Archit. News* **1990**, *18*, 239–248. [CrossRef]
31. Kumar, S.; Hughes, C.J.; Nguyen, A. Carbon: Architectural Support for Fine-grained Parallelism on Chip Multiprocessors. *SIGARCH Comput. Archit. News* **2007**, *35*, 162–173. [CrossRef]
32. Gordon, M.I.; Thies, W.; Amarasinghe, S. Exploiting Coarse-grained Task, Data, and Pipeline Parallelism in Stream Programs. In Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA, 21–25 October 2006; Volume 41, pp. 151–162.
33. Keckler, S.W.; Dally, W.J.; Khailany, B.; Garland, M.; Glasco, D. GPUs and the Future of Parallel Computing. *IEEE Micro* **2011**, *31*, 7–17. [CrossRef]
34. Ibrahim, K.Z.; Bodin, F.; Pène, O. Fine-grained parallelization of lattice QCD kernel routine on GPUs. *J. Parallel Distrib. Comput.* **2008**, *68*, 1350–1359. [CrossRef]
35. Di, P.; Ye, D.; Su, Y.; Sui, Y.; Xue, J. Automatic Parallelization of Tiled Loop Nests with Enhanced Fine-Grained Parallelism on GPUs. In Proceedings of the 2012 41st International Conference on Parallel Processing, Pittsburgh, PA, USA, 10–13 September 2012; pp. 350–359.
36. Li, K.B. ClustalW-MPI: ClustalW analysis using distributed and parallel computing. *Bioinformatics* **2003**, *19*, 1585–1586. [CrossRef] [PubMed]
37. Yang, L.; Misra, M. Coarse-Grained Parallel Algorithms for Multi-Dimensional Wavelet Transforms. *J. Supercomput.* **1998**, *12*, 99–118. [CrossRef]
38. Kundur, P.; Balu, N.J.; Lauby, M.G. *Power System Stability and Control*; McGraw-hill: New York, NY, USA, 1994; Volume 7.
39. Li, Z.; Yang, C.; Liu, K.; Hu, F.; Jin, B. Automatic scaling hadoop in the cloud for efficient process of big geospatial data. *ISPRS Int. J. Geo-Inf.* **2016**, *5*, 173. [CrossRef]
40. Crago, S.; Dunn, K.; Eads, P.; Hochstein, L.; Kang, D.I.; Kang, M.; Modium, D.; Singh, K.; Suh, J.; Walters, J.P. Heterogeneous Cloud Computing. In Proceedings of the 2011 IEEE International Conference on Cluster Computing, Austin, TX, USA, 26–30 September 2011; pp. 378–385. [CrossRef]
41. Nair, J.P.; Samuel, P. Analysis and Modeling of Resource Management Overhead in Hadoop YARN Clusters. In Proceedings of the 2017 IEEE 15th International Conference on Dependable, Autonomic and Secure Computing, 15th International Conference on Pervasive Intelligence and Computing, 3rd International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech), Orlando, FL, USA, 6–10 November 2017; pp. 957–964.