J Supercomput (2017) 73:4190–4205
DOI 10.1007/s11227-017-1987-9

Investigating Apache Hama: a bulk synchronous parallel computing framework

Kamran Siddique[1] · Zahid Akhtar[2] · Yangwoo Kim[1] · Young-Sik Jeong[1] · Edward J. Yoon[3]

Published online: 25 February 2017

Abstract The quantity of digital data is growing exponentially, and the task to efficiently process such massive data is becoming increasingly challenging. Recently, academia and industry have recognized the limitations of the predominate Hadoop framework in several application domains, such as complex algorithmic computation, graph, and streaming data. Unfortunately, this widely known map-shuffle-reduce paradigm has become a bottleneck to address the challenges of big data trends. The demand for research and development of novel massive computing frameworks is increasing rapidly, and systematic illustration, analysis, and highlights of potential research areas are vital and very much in demand by the researchers in the field. Therefore, we explore one of the emerging and promising distributed computing frameworks, Apache Hama. This is a top level project under the Apache Software Foundation and a pure bulk synchronous parallel model for processing massive scientific computations, e.g. graph, matrix, and network algorithms. The objectives of this contribution are twofold. First, we outline the current state of the art, distinguish the challenges, and frame some research directions for researchers and application developers. Second, we present real-world use cases of Apache Hama to illustrate its potential specifically to the industrial community.

Keywords Apache Hama · Bsp · Bulk synchronous parallel · Distributed computing · Mapreduce · Hadoop

✉ Yangwoo Kim
ywkim@dongguk.edu

1 Dongguk University, Seoul, South Korea

2 University of Quebec, Montreal, Canada

3 Samsung Electronics, Seoul, South Korea

1 Introduction

In recent years, there has been unprecedented data growth as information is continuously and rapidly generated from large internet sites, scientific experiments, government records, and sensors networks. The terminologies ‘big data’ and ‘large scale data’ were introduced to identify data that cannot be captured, curated, managed, or processed by traditional tools in a reasonable timeframe [1]. Big data or large scale data has four main characteristics: (i) very large data volume; (ii) data cannot be structured into regular database tables; (iii) data is produced at great velocity and must be captured and processed rapidly; (iv) low value density, large volumes data must be processed to extract the desired information. The quantity of data being generated every day is growing exponentially without apparent end, and it is one of the largest technological challenges in computing systems research to provide mechanisms for storage, information retrieval, and manipulation of these massive datasets. A rich set of tools have been developed to handle these huge volumes of data. The sheer volume of data requires significant computing power and space for processing. Generally, specialized hardware, such as super computers, to process the data is economically infeasible. Thus, particular frameworks such as parallel and distributed computing have been adopted as an economical alternative to provide the required computing power and resources. However, these large hardware clusters impose further challenges to traditional high-end hardware and processing, and it is vital to devise successful big data frameworks specifically addressing these challenges [2–4]. The requirements for research and development of massive computing frameworks are increasing tremendously, and emerging frameworks with sound technical and research potential could be major assets in the field.

Therefore, we present Apache Hama^{TM} [5], a pure bulk synchronous parallel (BSP) massive computing framework established in 2012, inspired by Google’s Pregel and DistBelief.

Necessity is the mother of invention, and MapReduce/Hadoop is no exception. These giant frameworks have dominated big data for over a decade and the MapReduce and Hadoop labels have almost become a synonymous with big data. However, the situation has changed, and the predominant Hadoop is approaching its limits. Academia and industry have recently recognized the limitations of the Hadoop framework in several application domains, and acknowledged that it cannot provide a one-size-fits-all solution for large scale processing [6]. In particular, a static map-shuffle-reduce pipeline of Hadoop architecture has become a major performance bottleneck in computationally intensive applications, and some applications have been compelled to replace MapReduce with new technologies [7], e.g. Google has already replaced it with Dataflow [8]. Thus, there is great interest to tackle big data processing challenges with a new wave of promising frameworks, such as Apache Spark, Apache Giraph, and Apache Hama. In contrast to Spark and Giraph, Hama has not yet been widely adopted, although some of its key features and performance benchmarks are sufficient to show its potential [9].

Our contribution in this particular area is motivated by observation of current trends in large scale data processing and observing the search log regarding Apache Hama over several research platforms [10,11]. The main contributions of this article are:

- Present an emerging high performance computing framework,
- Investigate research directions in big data processing using Hama,
- Document Hama's significant progress,
- Provide a thorough analysis for practitioners and users to acquaint themselves with Hama, and
- Unleash Hama's potential by presenting real-world use cases.

2 Research methodology

A systematic literature review requires comprehensive and unbiased coverage of relevant literature sources. The objective is to provide a valid assessment of a research topic through the application of a reliable, rigorous, and scrutinized methodology.

2.1 Sources

We chose the following digital libraries sources:

- IEEExplore Digital Library,
- ACM Digital Library, and
- WILEY.

2.2 Search strings

The following keywords or search terms were used:

- Apache Hama,
- Hama,
- Hama BSP, and
- Hama bulk synchronous parallel.

2.3 Selection of studies

After obtaining the results from the digital libraries, each candidate study must be analyzed individually to assure relevance with Hama. We did not limit the publication year, since Hama was proposed in 2010, rather, we limited the search to research papers available online and written in English. To select or discard studies, inclusion and exclusion criteria were defined as shown in Table 1.

2.4 Stages of selection

The stages involved in the selection of papers are listed in Table 2.

Searches on the digital libraries were conducted on October, 2015 and updated in January, 2016. To ensure an unbiased approach in subjective preference, two

Table 1 Include and exclude criteria

Include criteria	Exclude criteria
Directly or indirectly related to Apache Hama	Studies that only mention introduction to Hama
Studies using Hama as an experimental tool	Studies that do not have the complete text available at source
Studies published in journals and/or conferences	Not written in English
Written in English	

Table 2 Studies selection stages

Stages	Description
Stage 1	Using the defined keywords, apply the search query and gather the results
Stage 2	Exclude invalid and duplicate papers
Stage 3	Exclude studies on the basis of abstract, introduction, and conclusion
Stage 4	Review the selected studies and assess the relevance with topic of interest

Table 3 Studies by year

Publication year	Journal			Conference			References
	IEEE	ACM	Wiley	IEEE	ACM	Wiley	
2015	0	0	1	2	0	0	[13–15]
2014	0	0	1	1	2	0	[16–19]
2013	0	0	0	2	1	0	[20–22]
2012	0	0	0	1	1	0	[23,24]
2011	0	0	0	1	0	0	[25]
2010	0	0	0	1	0	0	[26]

researchers extracted the data, while others scrutinized the extraction. The bibliography and citation tool Zotero [12] was used to manage all extracted studies. Table 3 presents the yearly distribution of the selected papers.

After completing selection, each selected study was analyzed and reviewed. The next section provides an in-depth, unbiased and critical analysis of Apache Hama formulated as an interrogation session, based on the research literature.

3 Interrogation session

3.1 What is Hama?

Hama, previously known as HAdoop MAtrix and short for Apache Hama, is a top level project of the Apache Software Foundation. Hama is a distributed computing framework based on the BSP programming model [27], which acts as a bridge between

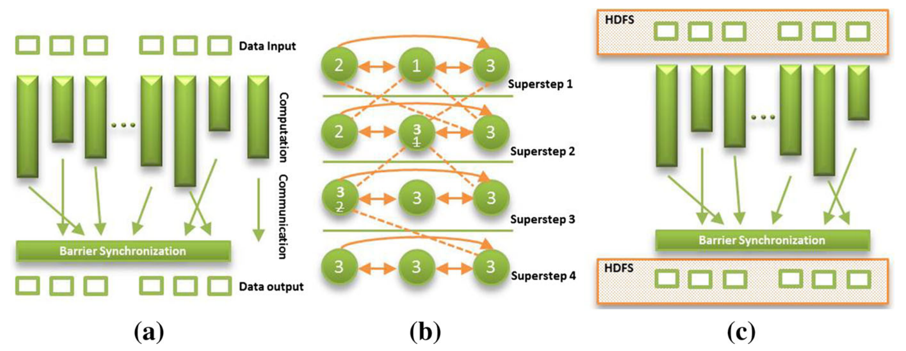

Fig. 1 Apache Hama follows bulk synchronous parallel (BSP) model **a** BSP model **b** A BSP computing example **c** Hama BSP framework

software and hardware for parallel computing. In BSP model, a parallel program runs across a set of virtual processors and executes as a sequence of parallel supersteps separated by barrier synchronization, as shown in Fig. 1a. This facilitates programmers by reducing the overhead of managing memory in local computations, manipulating global communications, and implementing efficient barrier synchronization. Figure 1b shows an example of computing the maximum vertex value using a BSP programming technique. The overall computation in a BSP based solution involves multiple supersteps and each superstep is composed of the following three ordered phases.

- Local computation
  - Each process performs local computation using local data values and issues communication requests for remote memory read and write operations.
- Global communication
  - Processes exchange their locally produced data according to the requests made during the local computation phase.
- Barrier synchronization
  - Ensures completion of all communication actions and makes previously exchanged data available to processes for use in the next superstep.

Apart from using the BSP model for computation and solving scientific problems based on graphs and matrices [28], Hama also provides deep learning packages for implementing scalable machine learning algorithms [29]. Hama was developed in Java, deployed on Hadoop Distributed File System (HDFS) as shown in Fig. 1c and can work seamlessly in any Hadoop environment.

3.2 What is Hama's architecture?

Apache Hama is based on a layered architecture and utilizes HDFS as the default file system. Hama's internal architecture differs from other known computational frameworks due to its underlying BSP based communication and synchronization mechanisms (see Fig. 2). It is based on a Master-Slave model consisting of three major components [30]:

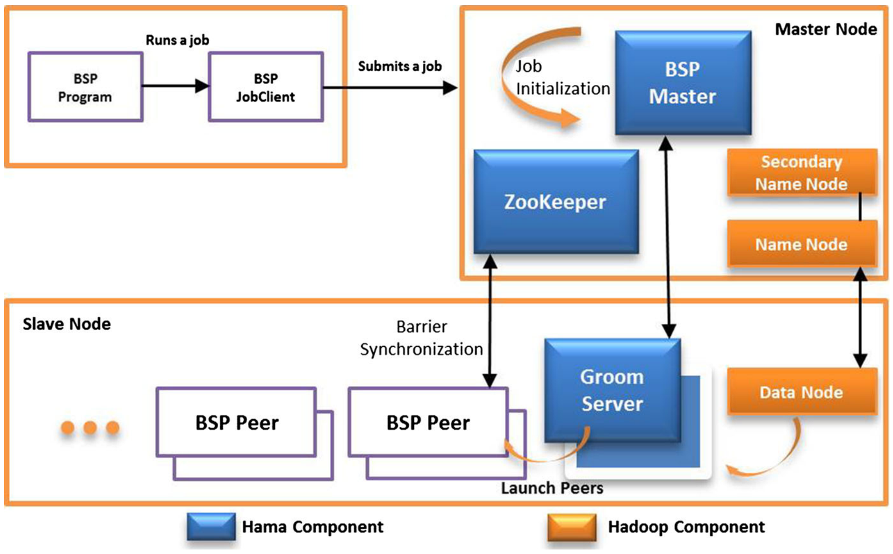

Fig. 2 Apache Hama architecture

1. BSP master performs the following functions:
   - Schedules jobs and assigns the tasks to a Groom Server,
   - Maintains the Groom Server status and job progress information,
   - Controls faults and supersteps in a cluster,
   - Distributes execution classes to its slaves, and
   - Provides a cluster control interface for users.
2. Groom Server, or simply Groom, acts as a slave component and it is responsible for running BSP peer tasks assigned by the BSP Master. It launches one or more BSP peer tasks and then each task acts as a worker task where the actual computation occurs. Each Groom sends a heartbeat to BSP master to report the status of processes and other metrics using periodical piggybacks. A Groom is flexible enough to run with any distributed storage in addition to HDFS. To achieve best performance, a Groom and a data node should run on the same physical node.
3. Zookeeper or synchronization component provides efficient barrier synchronization of the BSP peer tasks. Zookeeper and BSP Master execute on the same master node due to central barrier synchronization.

Figure 2 illustrates the Hama core architecture. BSP Program runs a job and BSP JobClient establishes the communication channel with BSP Master using Hadoop RPC framework. This component first partitions the input and then stores the chunks to HDFS before a new job is submitted to the BSP Master. It is executed locally, and periodically sends status updates including memory usage statistics for each process and superstep count. Each time the BSP Master receives a heartbeat message, it updates the Groom Server status. Using the updated status, the BSP Master efficiently assigns tasks to idle Grooms and returns a heartbeat response containing the set of actions to perform. Once a task is assigned, a Groom Server continues its execution until the last

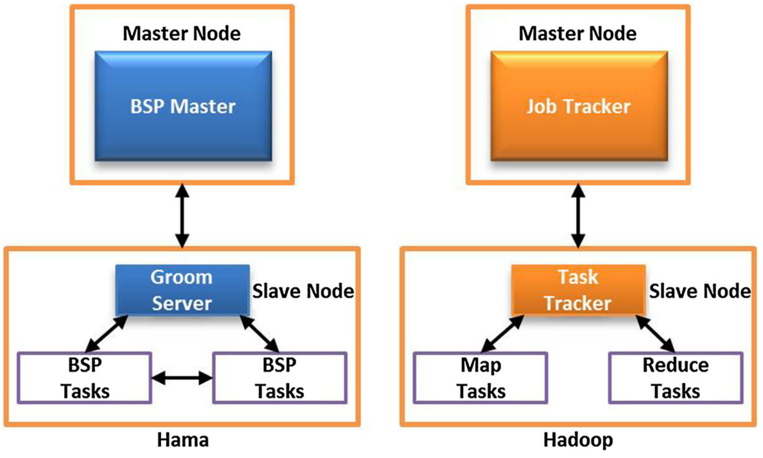

Fig. 3 Comparison between Hama and Hadoop architectures

superstep is executed. Upon task failure, it is marked failed and gets killed. During the whole execution life cycle, ZooKeeper manages efficient barrier synchronization of the processes.

3.3 How is Hama different from other big data frameworks?

Hama is a pure BSP model inspired by Google Pregel. Hama is more focused towards processing complex computation intensive tasks rather data intensive tasks, which makes it different from other frameworks. Despite Hama's similar architecture with Apache Hadoop and the inspiration from Google Pregel, it has significant differences. It aims to provide a more general purpose framework than Pregel and Apache Hadoop, supporting massive scientific computations such as matrix, graph, machine learning, business intelligence, and network algorithms. It is not restricted to graph processing, but also provides a full set of primitives that allows creation of generic BSP applications. The main differences between Hama and Hadoop architecture are illustrated in Fig. 3. In Hama, BSP tasks can communicate with each other, whereas communication between Map and Reduce tasks is forbidden in Hadoop. The MapReduce model also allows communication between the tasks only through the persistence of data on the disk, because the model enforces all Map tasks to complete execution before execution of any Reduce task. In contrast, Hama provides direct message exchange for the BSP tasks, which leads to better efficiency as I/O operation overheads are avoided.

3.4 What are the strengths of Hama?

Hama is a pure BSP programming model, and its versatile processing techniques and diverse application domains distinguish it from other distributed computing frame-

Table 4 Key attribute comparison of Hama with some well adopted frameworks

Framework	Processing Method	Application Domains			
		Graph	Streaming	Machine Learning	Incremental Learning
Hama	DMV	✓	✓	✓	✓
Hadoop	D	✓	✓	✓	✗
Spark	D	✓	✓	✓	✗
Giraph	V	✓	✗	✗	✗

Processing technique: *D* Directed Acyclic Graph; *M* Matrix; *V* Vertex-centric

Table 5 Some powerful BSP primitive operations implemented in Hama

Operation	Description
getPeerName()	Returns the name of the peer
getSupestepCount()	Returns the count of supersteps
send()	Sends a message to another peer
getCurrentMessage()	Returns a message received from the peer
getNumCurrentMessages()	Returns the number of received messages from the peer
clear()	Clears the entries of all queues
write()	Writes a key/value pair to the output interface
sync()	Performs barrier synchronization operations

works, as shown in Table 4. The robust BSP model also enables Hama to avoid conflicts and deadlines during communication at the largest scale. Unlike other frameworks, Hama provides BSP primitives to allow researchers and developers to operate at a lower level, rather than relying on limited graph processing APIs. Thus, Hama follows the standard BSP library at large scale.

Table 5 lists some powerful primitive operations or functions, and Fig. 4 demonstrates their use with an example. These simple, small, and flexible primitives have been used to create significant projects, such as Apache Horn, a neuron centric machine learning package.

Hama was primarily proposed to be used with java, but later enabled programmers to write in C++ [31]. An explicit support to the message passing interface is another advantage. Hama is flexible enough to be used with any distributed file system in addition to HDFS, and supports general purpose computing on graphics processing units (GPGPU) acceleration.

However, no system comes without limitations, whether they are related to basic architecture, development, or performance. In the case of Hama, BSP Master is a single point of failure and the application will stop if it dies. Despite great advances in graph processing performance, the manipulation functions remain somewhat limited. The graph partitioning algorithm needs to be customized, which will help avoid communication overhead between nodes.

```
public class MyBSP extends BSP<Null, Null, int, Text, Text>
       /* input keys: Null, i.e., no input available */
       public void bsp(BSPPeer<Null, Null, int, Text, Text> peer) throws
       IOException, SynException, InterruptedException
       /* performs local computations */
       peer.send(masterTask, message); /*messages sent to master task */
       peer.sync();                    /*initiates barrier synchronization
                                       & sends all the available messages
                                       to the corresponding remote peers */
               IF peer.getPeerName() = masterTask
               Text := msg_recvd;
               Do
               peer.write(int(0), msg_recvd); /*writes the output */
               while msg_recvd = peer.getCurrentMessage()!= Null;
               END_IF
       END_bsp()
END_MyBSP
```

Fig. 4 pseudo code demonstrating some BSP operations

3.5 Which application domains would Hama be the most suitable choice?

Hama is a general purpose solution for large scale computing and may be more suitable for intensive iterative applications. Hama outperforms MapReduce frameworks [25,26,32,33] in such application domains because it avoids MapReduce processing overheads, such as sorting, shuffling, reducing the vertices, etc. MapReduce inherits this overhead in each iteration, and there are at least millions of iterations. Hama provides a message passing interface and each BSP superstep is faster than a full job execution in the MapReduce framework, such as Hadoop.

3.6 Can Hama be applied to deep learning frameworks?

Recent advances in deep learning could potentially allow machine learning algorithms to extract discriminative information from big data without labor intensive feature engineering. Although a few very large companies, such as Google and Microsoft, have developed distributed deep learning systems, these are closed source systems. However, Apache Hama provides open source distributed training of artificial neural networks (ANNs) using the BSP computing engine. Two types of components are involved in the training procedure: (i) master task (merge the model, update information, and send model update information to all groom tasks); and (ii) groom task (calculate the weight updates according to the training data). Hama's ANN is currently data parallel only. Research is underway to support data and model parallelism.

3.7 Can Hama outperform MapReduce based frameworks?

Ting et al. [25] proposed a system architecture based on the concept of cloud computing. The proposed system is not only a data warehousing system but also a social networks analysis (SNA) engine that can be used to perform several SNAs with high performance. A performance comparison between MapReduce and Hama was also demonstrated, running a crawling program on different uniform resource locators (URLs). The authors highlighted a MapReduce weakness in mathematical graph processing, and showed Hama had a superior computation engine [26,32,33]. Since MapReduce is originally a function, it must pass the graph state from one step to another, which causes low efficiency during graphic algorithm processing. The experiment was performed to retrieve and store data from 100, 1000, 2000 and 10,000 different URLs. With increasing number of URLs, Hama provided better performance than MapReduce, accomplishing the task in less time.

Clustering techniques are important unsupervised learning methods, and K-means is one of most widely used and fast clustering algorithms in data mining. However, it has various issues, particularly for high dimensional and/or large data sets. MapReduce and BSP models are the prominent candidates to address these problems, used by Hadoop and Hama respectively. Although the MapReduce model is reliable and fault tolerant, it does not consider the distribution of input splits in the distributed file system during allocation of input files for map tasks, which results in increased data transfer overheads when running map tasks [34]. BSP overcomes this drawback by running the algorithms entirely within the cluster memory. Golghate and Shende [32] evaluated the performance of parallel K-means clustering using MapReduce and BSP techniques, and showed that BSP programming completed text clustering in a relatively short period, i.e., Hama was much faster than MapReduce because it does not have to submit a new job for each corresponding computation. The BSP superstep is computationally inexpensive compared to a MapReduce job, which provides better performance.

The performance of any distributed computing frameworks impacts several important applications. Li and Xu [33] analyzed Hama and Hadoop performance for efficiency and accuracy of their iterative algorithms. In particular, they selected the Monte Carlo calculation of Pi [35] since the algorithm has fixed execution results and is suitable for parallelization. The experiments in the study were conducted on a 4 node Hadoop cluster, and the Monte Carlo algorithm was implemented in Hama and Hadoop using the same software and hardware environment. They showed that Hama was superior to Hadoop, because Hama does not require excessive read and write operations, whereas Hadoop suffers from the large number of I/O operations.

3.8 Is there any framework that claims to outperform Hama?

Some big data processing solutions have demonstrated they outperform Hama. For example, Wang et al. [20] proposed a BSP based system (BC-BSP+) to process large graphs iteratively. BC-BSP+ supports a flexible configuration, efficient buffer disk management and multiple graph partition strategies as key features. The solution exploits the concept of virtual memory to store graph data and intermediate messages, dividing the JVM heap space into three parts to cache graph data objects and messages. The communication overhead is similarly reduced by introducing three graph partition strategies: i) randomized hash partitioning, ii) balanced hash partitioning, and iii) vertex-cut based on the range partition. The first scheme is simple while the others are used for load balancing and efficient utilization of the graph locality, respectively.

Thus, the proposed solution demonstrated better performance than Hama and Giraph as well.

Similarly, Ho et al. [21] proposed Kylin, also based on the BSP model, to provide efficient graph processing. The authors highlighted some limitations of existing BSP based frameworks: i) if message passing among workers is not handled properly the synchronization phase becomes the performance bottleneck ii) existing systems suffer from lack of data locality due to using the hashing technique for distributed data iii) most existing systems use the HDFS storage system, which does not provide a data scheme or API for data management, so users need to implement various input formats. It also does not support any indexing mechanism. Kylin addressed these issues by introducing three techniques: pull messaging, lazy vertex loading, and vertex-weighted partitioning. The proposed solution demonstrated up to 5 times better performance than that of Hama.

Chen et al. [18] identified issues with existing graph processing frameworks, and compared Hama and Cyclops, a vertex-oriented graph processing framework. They proposed a key abstraction technique, distributed immutable view (DIV), which provided a shared memory abstraction for graph algorithms. The evaluation tests were performed on typical pull and push graph algorithms, using datasets with 48 nodes. Their proposed solution outperformed Hama considerably. For pull-mode algorithms such as PageRank, the performance enhancement was due to the elimination of redundant computation and message passing for converged vertices, and exploiting the parallelism and locality of message passing via contention elimination. For push-mode algorithms such as single source shortest path (SSSP), the performance enhancement was due to optimized message passing and efficient vertex access through shared memory. Experiments were also performed to measure memory usage, computation, and communication efficiency, and Cyclops also showed better performance than Hama.

Zhou et al. [16] proposed Arbor, a large scale graph data processing system focused on efficient space utilization, fast processing, and increased parallelism. Arbor incorporates the advantages of hyper-graph and simple-graph by the proposed extended simple graphs (ESGs), which reduced data representation complexity. A control message mechanism was proposed to mitigate the expensive synchronization operations during data iterations, and two optimization strategies, check before sending (CBS) and avoid unnecessary messages (AUM), were also introduced to improve overall system performance. Experiments were performed on two applications, PageRank and SSSP, using average response time as the key metric for performance comparison. The performance evaluation of the SSSP algorithm found that Arbor's average response time was at least 59% lower than Hama, and Arbor's performance increased with increasing number of vertexes. Arbor also outperformed Hama significantly in the PageRank evaluation, with reduced average response time of at least 17% in all scenarios.

Luo et al. [17] discuss drawbacks of the BSP graph partitioning mechanism used in Hama. The authors studied and analyzed Hama source code (version 0.6.3) which does not use any special algorithm for graph partitioning. Although BSP provides better performance than MapReduce for parallel graph mining, network communication between nodes is still the performance bottleneck in this model because it is slower than memory or disk I/O. The number of edges in the partition should be higher than the number of cross partition edges. An efficient partition algorithm can minimize the number of cross partition edges and reduce the communication overhead between computers. The authors suggested that Hama's graph partition strategy may have a negative performance impact because of the unnecessary communication between nodes.

4 Use cases

We present some real-world use cases of Apache Hama to illustrate its current state and potential at industrial scale.

4.1 Hama at Sogou

4.1.1 Summary

A well-known Chinese search engine, Sogou [36], successfully deployed Hama to compute SiteRank.

4.1.2 The challenge

Efficient searching tools have been a boon to web users. With continuous growth of online data, querying and retrieving relevant information is becoming increasingly challenging. The classic PageRank algorithm still provides tremendous service to facilitate web users provided the response time and result relevancy is significant. This requires an efficient data processing engine to accomplish the task.

4.1.3 Solution

PageRank is one of the methods used by search engines to determine a page's relevance or importance. When a user enters a search query, the search engine's number one goal is to return results that are highly relevant in a timely manner. Generally, a search engine considers 200+ factors to determine relevance, and PageRank is the most common algorithm employed. On similar patterns, the Sogou search engine runs PageRank algorithm on 7200 core Hama clusters, over a dataset exceeding 400 GB, containing 600 M vertices and 6 Billion edges.

4.2 Hama at Korea telecom

4.2.1 Summary

A large mobile carrier, Korea Telecom [37], used Hama to monitor its network traffic and capture issues as they arose.

4.2.2 The Challenge

Korea telecom, similar to most large telecom operators, monitors their traffic to analyze and identify issues related to network outages or changed network loads that need to be managed. Prior analysis and prediction of outages or workload changes offers advantages in terms of lower operational costs and better service. However, abrupt network issues and workload changes arise irregularly, and often unpredicted. Therefore, prediction quality and speed of reaction to changes in bandwidth are critical. To address such issues, the company needs to

- Analyze historical network data,
- Analyze sensor and related data, and
- Provide real-time services and reports.

4.2.3 Solution

Korea Telecom deployed Hama, and achieved the following:

- Processing network data in real-time and identified problems,
- Identified patterns in the data to forecast problems,
- Maintained aggregate statistics and related data to immediately report patterns of interest, and
- Efficient storage system to keep historical records of network data.

4.3 Hama at Samsung electronics

4.3.1 Summary

Samsung Electronics [38] employs Hama for real-time and large scale processing.

4.3.2 The challenge

Samsung Electronics has "brick and mortar" stores as well as a huge medical services system, where it has recently applied deep learning technology to assist doctors in diagnosing diseases, such as breast tumors, etc. The company primarily wants to facilitate its customers providing purchasing recommendations in real-time. To achieve this, they must integrate data and information from different sources, such as social media, point of sale, historical web logs, inventory, customer relationship management (CRM), and real-time web activity.

4.3.3 Solution

The solution was implemented using Hama in a series of steps:

- Analyze the data sources to determine customer likes, dislikes, and buying behavior using logged activity;

- Integrate the resulting information with CRM, inventory, and stock information; and
- Instant recommendations provided for products the customer may be interested in purchasing.

5 Conclusions and future directions

In the digital and computing world, information is generated and collected at an exponentially expanding rate. Therefore, big data concepts have received much attention from both academia and the IT industry. Big data concepts and infrastructures are becoming the standard approach for many applications. Apache Hama is a relatively new framework, that is rapidly gaining momentum.

This paper highlighted and explored Hama as a potential research area for big data processing. While research on big data processing using Apache Hama is in its early stages, it is essential to identify future directions from a critical analysis. Therefore, we provided a significant and focused interrogation session for Hama, identifying promising areas that justify further exploration and development, e.g. specialized graph partitioning algorithms, load balancing, optimization of memory usage, and fault tolerance mechanisms.

We also presented some real-world use cases that show Hama's current position in both academia and industry. To the best of our knowledge, this Apache Hama update is missing from the current literature, and we hope this article will assist researchers to focus their research time and career in this particular area.

In the future, we intend to conduct several benchmark evaluations comparing Apache Hama performance with other massive computing frameworks. This will further help forecast Apache Hama's future and open new doors for interested researchers.

Acknowledgements This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the University Information Technology Research Center support program (IITP-2016-R2720-16-0004 and IITP-2016-H8501-16-1015) supervised by the IITP (Institute for Information & Communications Technology Promotion).

References

1. Anagnostopoulos I, Zeadally S, Exposito E (2016) Handling big data: research challenges and future directions. J Supercomput 72(4):1494–1516. doi:10.1007/s11227-016-1677-z
2. Gebara FH, Hofstee HP, Nowka KJ (2015) Second-generation big data systems. IEEE Comput 48(1):36–41. doi:10.1109/MC.2015.25
3. Yu N, Yu Z, Li B, Gu F, Pan Y (2016) A comprehensive review of emerging computational methods for gene identification. J Inf Process Syst 12(1):1–34. doi:10.3745/JIPS.04.0023
4. Kolici V, Herrero A, Xhafa F (2014) On the performance of oracle grid engine queuing system for computing intensive applications. J Inf Process Syst 10(4):491–502. doi:10.3745/JIPS.01.0004
5. Apache Hama. https://hama.apache.org/. Accessed 25 March 2016
6. Kalavri V, Vlassov V (2013) MapReduce limitations, optimizations and open issues. In: The IEEE 12th International Conference on Trust, Security and Privacy in Computing and Communications, pp 1031–1038
7. Fortune. http://fortune.com/2015/09/09/cloudera-spark-mapreduce/. Accessed 25 March 2016

8. InformationWeek. http://www.informationweek.com/cloud/software-as-a-service/google-i-o-hello-dataflow-goodbye-mapreduce/d/d-id/1278917. Accessed 25 March 2016
9. Elser B, Montresor A (2013) An evaluation study of BigData frameworks for graph processing. In: IEEE Big Data pp 60–67
10. Apache Apache Software Foundation blogging in action. https://blogs.apache.org/Hama/. Accessed 10 January 2016
11. Mailing list archives. https://hama.apache.org/mail-lists.html. Accessed 10 January 2016
12. Zotero. https://www.zotero.org/. Accessed 15 October 2015
13. Friedman R, Portnoy A (2015) A generic decentralized trust management framework. Softw Pract Exp 45(4):435–454. doi:10.1002/spe.2226
14. Zhang X, Wang R, Chen X, Wang J, Lukasiewicz T, Han D (2015) Achieving up to zero communication delay in BSP based graph processing via vertex categorization. In: International Conference on Networking, Architecture, and Storage, IEEE, Boston, pp 112–121. doi:10.1109/NAS.2015.7255213
15. Ratnaparkhi AA, Pilli E, Joshi RC (2015) Scaling GMM expectation maximization algorithm using bulk synchronous parallel approach. In: International Conference on Green Computing and Internet of Things, IEEE, Noida, pp 558–562. doi:10.1109/ICGCIoT.2015.7380527
16. Zhou W, Han J, Gao Y, Xu Z (2016) An efficient graph data processing system for large-scale social network service applications. Concurr Comput 28(3):729–747. doi:10.1002/cpe.3393
17. Luo S, Liu L, Wang H, Wu B, Liu Y (2014) Implementation of a parallel graph partitioning algorithm to speed up BSP computing. In: The 11th International Conference on Fuzzy Systems and Knowledge Discovery. IEEE, China, pp 740–744
18. Chen R, Ding X, Wang P, Chen H, Zang B, Guan H (2014) Computation and communication efficient graph processing with distributed immutable view. In: The 23rd International ACM Symposium on High Performance Parallel and Distributed Computing. Vancouver, Canada, pp 215–226
19. McColl R, Ediger D, Poovey J, Campbell D, Bader DA (2014) A performance evaluation of open source graph databases. In: The Proceedings of the First Workshop on Parallel Programming for Analytics Applications. Orlando, Florida, pp 11–17
20. Wang Z, Bao Y, Gu Y, Leng F, Yu G, Deng C, Guo L (2013) A BSP based parallel iterative processing system with multiple partition strategies for big graphs. In: IEEE International Congress on Big Data, CA, pp 173–180
21. Ho LY, Li TH, Wu JJ, Liu P (2013) Kylin: an efficient and scalable graph data processing system. In: IEEE International Conference on Big Data, CA, USA, pp 193–198
22. Khayyat Z, Awaraz K, Alonaziz A, Jamjoomy H, Williamsy D, Kalnis P (2013) Mizan: a system for dynamic load balancing in large-scale graph processing. In: Proceedings of the 8th ACM European Conference on Computer Systems. Czech Republic, Prague, pp 169–182
23. Zhang J, Ge S (2012) A parallel algorithm to find overlapping community structure in directed and weighted complex networks. In: 2nd International Conference on Instrumentation and Measurement, Computer, Communication and Control, IEEE, Harbin City, Heilongjiang, China, pp 1561–1564. doi:10.1109/IMCCC.2012.364
24. Chen R, Weng X, He B, Yang M, Choi B, Li X (2012) Improving large graph processing on partitioned graphs in the cloud. In: ACM Symposium on Cloud Computing, San Jose, CA. doi:10.1145/2391229.2391232
25. Ting IH, Lin CH, Wang CS (2011) Constructing a cloud computing based social networks data warehousing and analyzing system. In: International Conference on Advances in Social Networks Analysis and Mining. IEEE, Kaohsiung, Taiwan, pp 735–740
26. Seo S, Yoon EJ, Kim J, Jin S, Kim JS, Maeng S (2010) HAMA: an efficient matrix computation with the MapReduce framework. In: Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom). Greece, Athens, pp 721–726
27. Valiant LG (1990) A bridging model for parallel computation. Commun ACM 33(8):103–111
28. Hama Graph Tutorial. https://hama.apache.org/hama_graph_tutorial.html. Accessed 10 January 2016
29. Apache Horn. http://horn.incubator.apache.org/index.html. Accessed 10 January 2016
30. Apache Hama Design Document V0.6. http://people.apache.org/~tjungblut/downloads/hamadocs/ApacheHamaDesign_06.pdf. Accessed 20 December 2015
31. Apache Hama Pipes Development Repository. https://github.com/millecker/hama-0.5.0-gpu. Accessed 10 January 2016
32. Golghate AA, Shende SW (2014) Parallel K-means clustering based on hadoop and hama. Int J Comput Technol 1(3):33–37

33. Li S, Xu B (2015) Performance comparison between hama and hadoop. Int J Database Theory Appl 8(3):77–84
34. Jin S, Yang S, Jia Y (2012) Optimization of task assignment strategy for map-reduce. In: 2nd International Conference on Computer Science and Network Technology. Changchun, China, pp 57–61
35. Module for Monte Carlo Pi. http://mathfaculty.fullerton.edu/mathews/n2003/montecarlopimod.html. Accessed 10 January 2016
36. Sogou Inc. https://www.sogou.com. Accessed 10 January 2016
37. KT Corporation. https://www.kt.com/eng. Accessed 10 January 2016
38. Samsung Electronics. https://www.samsung.com. Accessed 10 January 2016