Comparing Data Processing Frameworks for Scalable Clustering

Sharanjit Kaur
Department of Computer Sc
Acharya Narendra Dev College
University of Delhi
Delhi, India
skaur@cs.du.ac.in

Rakhi Saxena
Department of Computer Sc
Deshbandhu College
University of Delhi
Delhi, India
rsaxena@db.du.ac.in

Dhriti Khanna
Department of Computer Sc
Acharya Narendra Dev College
University of Delhi
Delhi, India
dhriti0610@gmail.com

Vasudha Bhatnagar
Department of Computer Sc
South Asian University
Delhi, India
vbhatnagar@cs.sau.ac.in

Abstract

Recent advances in the development of data parallel platforms have provided a significant thrust to the development of scalable data mining algorithms for analyzing massive data sets that are now commonplace. Scalable clustering is a common data mining task that finds consequential applications in astronomy, biology, social network analysis and commercial domains. The variety of platforms available for implementation of algorithmic solutions poses a dilemma of choice for researchers and practitioners, alike.

This paper presents a case study for comparison of two open source data parallel frameworks for implementation of a scalable clustering algorithm. Ascension of Map-Reduce framework to the top slot in less than a decade is attributed to its ease of programming, robustness and capability to perform automatic load balancing. Recently proposed Nephele-PACT framework is a serious contender, because of its design that offers optimized execution of complex data flows, prevalent in *Big Data Analytics*. Both frameworks use HDFS as the underlying distributed file system, hence the comparison is confined to the execution engines only. Experimentation on real and synthetic data reveals the better scalability of Nephele-PACT framework.

Keywords: MapReduce, PACT, Scalable Clustering, HDFS

1 Introduction

Developing scalable algorithms for automated or semiautomated analysis is a compulsive need of the analytics community, owing to the rapidly growing number of massive data sets. Advances in digital sensors, communications and storage are the significant causal factors for the current data deluge. Search engine companies like Yahoo, Google, and Microsoft are encashing on popularity and advertisements by capturing freely available *big* data on the web and providing useful services to masses (Bryant 2011). However, it is imprudent to process such hugely abundant data on a single machine.

Deployment of cluster computing frameworks, which aggregate computational power, main memory and I/O bandwidth of shared-nothing commodity machines, has emerged as the most popular solution (Pavlo et al. 2009; Rajaraman and Ullman 2012). Google was the earliest propounder of this technology for carrying out analytics on *Big Data*. The endeavor was founded on a solid proposal for a data parallel programming model and accompanying framework called MapReduce (MR) (Dean and Ghemawat 2004), designed to execute over Google File System (Ghemawat, Gobioff, and Leung 2003). Apache open source response to the two Google technologies is Hadoop-MR and Hadoop Distributed File System (HDFS), respectively. The spectacular rise in popularity of MR framework for data parallel computing on large shared-nothing commodity clusters is attributed to its ease of programming, robustness and ability to perform automatic load balancing. Programmers with little experience in parallel programming and distributed systems, can quickly develop data parallel programs for execution on a large cluster using abstract functions *Map* and *Reduce* (Dean and Ghemawat 2004).

MapReduce (MR) has inspired more than a dozen variants and extensions including Clustera, Hyracks, Spark, Dyrad, Pregel, MapReduce Merge, SCOPE, Nephele-PACT etc.. Some of these cater to specific needs of applications like processing of iterations, joins, graphs and streams. Others are designed to offer significant advancements in processing of complex data flows. An extensive survey of MR variants and its extensions for large scale data processing is available in (Sakr, Liu, and Fayoumi 2013).

Existing data parallel programming platforms can be categorized into two distinct classes. The first class comprises the likes of MR, which capture sequential data flows and execute them efficiently, and the second class consists of platforms that possess the capability to capture, optimize and execute complex data flows using additional communication channels such as sockets, pipes or shared memory. MR, MR-Merge, etc. fall in the first category, while Dryad, Nephele-PACT, Hyracks etc. fall in the second category.

Availability of several data parallel platforms bestows the

luxury of choice to practitioners and researchers. Depending on the distinguishing characteristics of the algorithm, a practitioner is keen to choose a platform that delivers the best return on investment. A researcher on the other hand is interested in developing algorithms that bring out the potential of the underlying platform to the best possible extent. A comparative study of data parallel, distributed computing platforms aids in matching the algorithm and the platform.

In this paper, we compare Hadoop-MR and Nephele-PACT frameworks, which are open source representatives of the two categories mentioned earlier. We choose a scalable stream clustering algorithm (ExCC algorithm (Bhatnagar, Kaur, and Chakravarthy 2013)) as the subject of the case study, and describe in detail the design distinctions for the two platforms. The choice is motivated by two reasons. Firstly, clustering is one the most popular descriptive analytic techniques used for mining voluminous data sets. Secondly, clustering is representative of many practical data analytics tasks that require complex cascades of MR jobs rather than simple two step data parallel processing of MR. We decided to use ExCC algorithm because of its incremental nature and its capability to deliver recent, exclusive and complete clustering. Additionally, the algorithm possesses the ability to cluster data with mixed (numerical and categorical) attributes, commonly analyzed in several real life applications.

ExCC uses a *grid* structure as synopsis to summarize the data stream. Implemented as in-core *trie*, the grid discretizes data space into hyper-cuboidal regions called *cells*. Each of these *cells* can be processed independent of each other in parallel. The algorithm is perfectly suited for data parallelism, and is therefore an apt choice for implementation on a data parallel distributed computing environment.

Though this study uses a grid-based clustering algorithm as a vehicle for comparison, the insights gained highlight significant distinctions between the two frameworks and can be useful for selecting the better one for design/implementation of scalable clustering algorithms in general. Section 2 describes related work. Section 3 describes the design and implementation of the algorithm, bringing out distinctions that arise due to differences in two frameworks. The performance comparison of the two frameworks is given in Section 4 and conclusion in Section 5.

2 Related Work

In this section, we begin with a brief description of the Hadoop-MapReduce and Nephele-PACT frameworks, followed by a description of the ExCC algorithm.

2.1 Hadoop-MapReduce Framework

Hadoop-MapReduce is a data centric distributed framework that expresses simple computations as MapReduce jobs (Dean and Ghemawat 2004). The framework splits input data into small chunks, which are processed in parallel, on different compute nodes, by invoking Map tasks to execute user-defined functions. Subsequently, the framework initiates a *Sort and Shuffle* phase to channelize data to parallely executing Reduce tasks for aggregation. Thus, the framework permits a sequential data flow with the Map phase feeding the Reduce phase.

Features	MapReduce	Nephele-PACT
Data Flow Supported	Sequential	Complex (DAG)
Basic Primitives	Map, Reduce	Map, Reduce, Match Cogroup, Cross
Support for Multiple inputs	No	Yes
Identity Mappers/Reducers	Often Required	Optional
Global counters	Available	Not Available
Distributed Cache	Available	Not Available
Custom partitioning/Sorting	Possible	Not Possibble
Execution Strategy	Fixed	Optimized
Fault Tolerance	Fully Suppported	Experimental Stage

Table 1: Feature-wise comparison of Mapreduce and Nephele-PACT frameworks (DAG:Directed Acyclic Graph)

2.2 Nephele-PACT Framework

The *Nephele-PACT Framework* offers *Parallelization Contracts* (PACTs), which are generalizations of Map and Reduce primitives of the MR framework. The framework is motivated by the need to handle complex data flows in a cluster environment (Alexandrov, A. et al. 2010; Battré, D. et al. 2010). Novel dual input primitives like *Co-group, Cross* and *Match* contracts are available to the programmer to negotiate complex data analytic tasks.

Unlike the fixed execution strategy in MR, the PACT compiler generates multiple execution plans and selects the optimal one. The execution plans are evaluated and stored as directed acyclic graphs (DAG). The vertices of the DAG are instances of PACTs and the edges denote data transportation mechanism (file/network/in-memory channel) between the PACTs. The selected DAG is executed by the Nephele execution engine in parallel on multiple nodes. Table 1 contrasts the differences between MapReduce and PACT frameworks.

2.3 ExCC: A Stream Clustering algorithm

The *ExCC (Exclusive and Complete Clustering) algorithm* is designed to generate a recent, exclusive and complete clustering scheme from potentially infinite data streams (Bhatnagar, Kaur, and Chakravarthy 2013). It uses an in-core grid data structure as synopsis, which is implemented as an in-core *trie* structure. Based on user-specified granularity, the grid discretizes bounded data space into hyper-cuboidal regions called *cells*. Each *cell* has a unique signature depending upon its location in the data space and maintains summarized statistics like number of data points, and logical time-stamps of first and last data points received in the corresponding data region.

The algorithm has two components that execute in sequence. The first component localizes the incoming points to the appropriate *cell* by discretizing it and extracting its *signature*. The cell statistics are updated appropriately with each incoming point in order to maintain up-to-date synopsis. Periodically, non-recent cells are pruned away from the grid so that old data do not over-shadow current structures and the grid does not outgrow the memory. Time stamps preserved in the *cells* are used for judging recency,

The second component performs clustering after identifying dense cells in the grid. Density of a cell (number of points in the region) is used as the evidence of structure subsistence in data. Density threshold is computed dynamically as a function of average number of points in the grid and data dimensionality. Subsequently, recent dense cells in the grid are coalesced using connected component analysis to discover clusters. In the final stage, only significant clusters are reported. Interested reader may refer to the details in (Bhatnagar, Kaur, and Chakravarthy 2013)

3 Data Parallel Clustering Algorithm

In this section, we describe the MR-design for the proposed scalable algorithm, followed by a detailed description of the implementational differences that arise due to the underlying differences in the Hadoop-MR and Nephele-PACT frameworks.

3.1 Data Parallel design of ExCC

The data parallel version of ExCC is an incremental algorithm, which can handle potentially infinite data in batches (increments).

Figure 1 show the blue print of the algorithm with implementational differences in the two models. Jobs 1 and 2 accomplish the tasks of the first component of ExCC, and Jobs 3 and 4 accomplish the tasks of the second component. Interestingly, the proposed design precludes the need of maintaining in-memory *trie* structure for synopsis. Instead, the synopsis is stored as $\langle Key, Value\rangle$ pairs of $\langle cell\text{-}signature|cell\text{-}statistics\rangle$. Storing synopsis on file system imparts true scalability to the algorithm.

The algorithm takes the current increment, previous synopsis and necessary parameters (data grid granularity and dimension details) as input. The functionality of each job is described below.

i. JOB 1: *Cell signature generation*: This job takes the input as time stamped incremental data set, where each record is a $\langle timestamp|datavalue\rangle$ pair. Each data value is discretized in parallel to localize it in a *cell*. At each compute node, cells with same signatures are *combined* (aggregated) and the cell statistics are updated. The output of this job is a sequence of $\langle cell\text{-}signature|cell\text{-}statistics\rangle$ pairs stored in file, representing the synopsis for the current data increment.

ii. JOB 2: *Synopsis maintenance*: This job merges previous synopsis with current synopsis and enforces data recency by updating cells with same signature in parallel. Recent cells are identified using the time-stamps from aggregated statistics of *cells*. The output is the updated synopsis as $\langle Key, Value\rangle$ pairs, similar to that of JOB 1. The updated synopsis is used for processing the future data increment and also as input to JOB 4.

iii. JOB 3: *Preparation for partitioning*: The objective of this job is to prepare for partitioning the synopsis by assessing the degree of parallelism for accomplishing clustering in the next job. Updated synopsis from JOB 2 is the input to this job. For a selected dimension, *cell* signature value in that dimension of all the cells are emitted in parallel, with dimension number as the key. Unique values are sorted and *stringified* and passed to the next JOB.

iv. JOB 4: *Significant clusters Generation* : The string received from JOB 3 is examined to figure out the number of partitions that can be clustered in parallel (See example below). Synopsis from JOB 2 is the input for this job too. In the Map phase, each cell in the synopsis is assigned to a partition by emitting the partition number as key and cell signature as value. All cells with same partition number are thus collected at the same compute node for the Reduce phase where Connected Component Analysis (CCA) is applied for clustering the received cells. CCA performs clustering by selecting an arbitrarily dense cell and progressively coalescing dense cells that are adjacent to it or any of the (cluster) member cells (Agrawal et al. 1998).

We clarify the tasks carried out by JOBs 3 and 4 in the following example.

Example 1 *Consider an updated synopsis with 5 cells [0,2,2] [1,2,3] [3,3,1] [0,3,1] [4,3,1]. Suppose dimension 1 is fixed for ascertaining degree of parallelism. JOB 3 emits values (0, 1, 3, 0, 4) and sorting unique values results into (0,1,3,4). JOB 4 finds that the sorted string has two contiguous value substrings viz. (0,1) and (3,4). Thus all cells with 0 or 1 on first dimension are placed in one partition and cells with values 3 or 4 are placed in the second partition. Thus, two preliminary partitions are identified as {[0,2,2] [0,3,1] [1,2,3]} and {[3,3,1] [4,3,1]} respectively, each of which is clustered in parallel.*

3.2 Implementational Distinctions

In this section, we highlight the implementational distinctions that arise due to the underlying differences in Hadoop-MR and Nephele-PACT platforms. For the sake of clarity, we refer MR version as ExCC-MR and PACT version as ExCC-P. Dashed blocks in Figure 1 reveal the implementational differences of two versions.

i. JOB 1: The job needs program parameters to be available at each node before the processing of current data increment. MR provides the distributed cache feature for broadcasting read-only meta data, wherein the cached data is copied to all nodes before launching a job. *However, initial experimentation with distributed cache in ExCC-MR revealed the overheads involved for distributing program parameters in this manner.* PACT does not provide distributed cache facility. Instead, use of Cross contract is recommended by the Nephele-PACT designers. *Again, initial experimentation found this approach inefficient for the job. The Cross contract was able to generate the signature value of only a single dimension at a time, a subsequent Reduce contract was required to compose the complete cell signature on all dimensions.*

However, both platforms offer provision for passing small data to the compute nodes as $\langle Key, Value\rangle$ pairs via the Job Configuration object. Hence program parameters are encoded as a string that is passed to all MAP tasks using this mechanism in both frameworks.

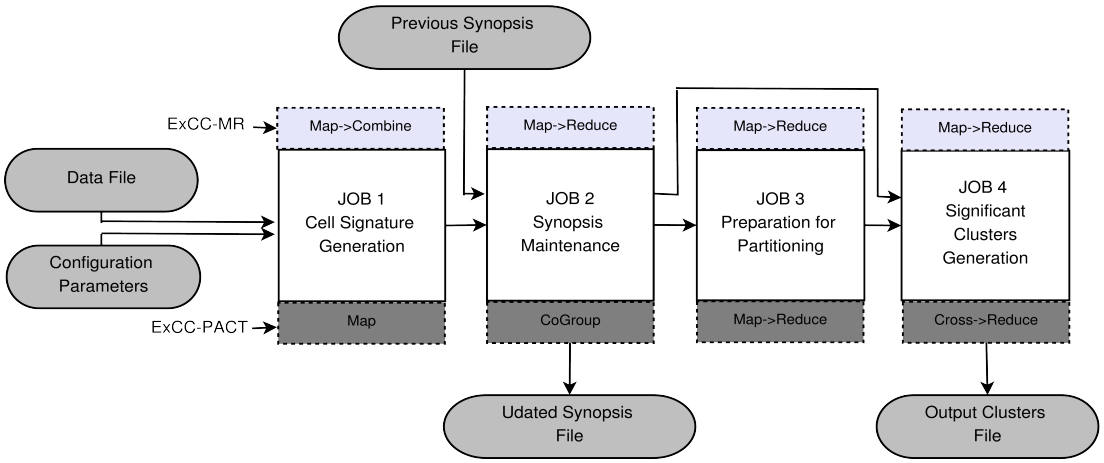


Figure 1: Four jobs identified for ExCC for distributed computing and its implementation in MapReduce and PACT

ExCC-MR uses a combiner for local aggregation of synopsis on each node. Since the job does not require any reduce functionality, ExCC-MR sets the number of reducers to 0. Lack of a Reducer in this job implied that ExCC-P could not aggregate synopsis locally since PACT provides the option of using a combiner only via a reduce task.

Hadoop, which supports pure MR programming model writes output in the HDFS. Consequently the current synopsis is written on the disk, requiring increased disk space and network bandwidth. The Nephele-PACT model is capable of supporting complex data flows through the use of network channels. ExCC-P uses this facility and pipelines the current synopsis to the next job saving on the I/O overhead (Section 4).

ii. JOB 2: This job entails merging the previous synopsis with the current synopsis. ExCC-MR employs an identity *mapper* to forward the two synopses residing on HDFS to the reducers. This causes a surge in the use of network bandwidth. Pipelining of the current synopsis directly to the *Co-Group* contract through the network channel by the map contract in ExCC-P precludes the need for identity mappers and optimizes resource usage. Accordingly, ExCC-P receives the current synopsis through the network channel and the previous synopsis from HDFS.

   The reducers in ExCC-MR aggregate cell statistics, determine recency and output only the recent cells. Similar actions are performed by *Co-Group* contract in ExCC-P, which hands over the output to *FileDataSinkContract* for storing the updated synopsis.

   Global counters provided by MapReduce framework permit aggregation across all maps and reducers to produce a grand total at the end of the job. This facility comes handy in ExCC-MR to count total cells and data points in the updated synopsis. These counters are used to determine density threshold in JOB 4. Since PACT lacks this facility, determination of these counts is deferred to JOB 3 in ExCC-P. This does not cause any overhead in ExCC-P in this particular algorithm, but may cause overheads or lead to complex designs of some analytics tasks.

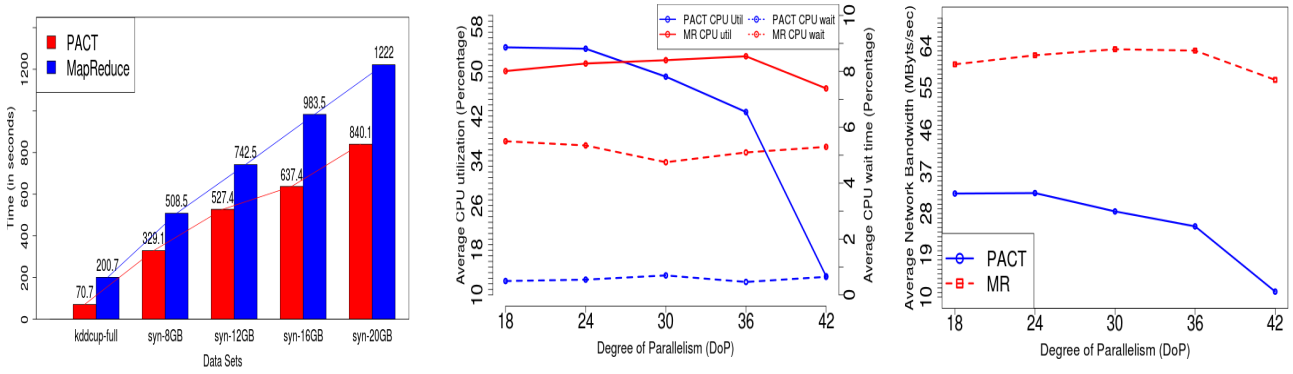
iii. JOB 3: Implementation of this job is similar in both frameworks, except for two distinctions. First, *Co-Group* contract in ExCC-P delivers the updated synopsis to this job through network channel instead of disk, thereby saving data transfer over network. Secondly, ExCC-P has to additionally count the cells and data points, which ExCC-MR had accomplished in the previous job.

   The output of JOB 3 is passed to JOB 4 as $\langle Key, Value \rangle$ pair in the Job Configuration object by ExCC-MR and is broadcast through the network channel by ExCC-P.

iv. JOB 4: This is the core clustering job. ExCC-MR reads the input from disk using a map task, while ExCC-P receives updated synopsis pipelined through the network from JOB 2 into a *Cross* contract. The map task in ExCC-MR and the *Cross* Contract in ExCC-P divides the cell pool into disjoint partitions. Both implementations derive the partition number from the sorted list of distinct granularity values of the pre-fixed dimension (See Example 1), received as memory input from JOB 3. The reducers in both implementations, retain the dense cells, apply CCA for clustering and report only significant clusters.

In summary, the major point of distinction is the use of network channel for passing data between contracts. As revealed in experiments, this leads to substantial saving in execution time because of dramatically reduced I/O. Absence of a distributed cache in the PACT programming model can be surmounted by passing data to distributed tasks via $\langle Key, Value \rangle$ pairs in the Job Configuration. However, the absence of Counters for global aggregation in PACT forces collection of all values to be aggregated into a single reducer. This would be a bottleneck when large number of values are to be aggregated and could cause the program to fail if the values would not fit into the memory of a single compute node.

As most analytics task involve complex data flows, which are unnatural to be modeled as two step sequential MR jobs, Nephele-PACT offers a framework for elegant design. Currently however, lack of fault tolerance is a serious limitation



(a) Total runtime (in seconds) with increasing dataset size

(b) Average CPU utilization and average CPU waiting time with increasing degree of parallelization

(c) Average network bandwidth usage with increasing degree of parallelization

Figure 2: Performance Comparison of ExCC-MR on Hadoop-MR and ExCC-P on Nephele-PACT

of Nephele[1].

4 Experiment Evaluation

We experimented on a 6-node Intel Xeon(R) machine cluster with Ubuntu Linux (version 13.04, kernel 3.8.0-30-generic) as operating system. Each node had E3-1240V2 @ 3.40GHz X 8 processors, 15.6GB of RAM, and available storage of 476GB. The test cluster consisted of a total of 48 cores and 1.06 TB storage. All machines were connected through regular 1 GBit/s Ethernet links.

Sun JVM version 1.7.0 was installed to run stable Hadoop version 0.20.2 and Stratosphere version 1.2. Default block size of HDFS (64 MB) was used to distribute data, with default replication factor. Ganglia (University of California, Berkeley 2006) was used to monitor real-time usage of resources of the cluster. The goal of the experiments was to compare performance of Hadoop MapReduce and Nephele PACT for the designed algorithm. In particular, we wanted to answer following two questions:

i. How do execution times scale with data size for the two frameworks?

ii. How do CPU utilization, waiting time and network bandwidth vary with varying degrees of parallelization in the two platforms?

Dataset description: We performed experiments to compare resource usage and run time scalability of ExCC-MR versus ExCC-P using one public dataset and four synthetic datasets described in Table 2. KDD dataset, with 34 numeric and 7 categorical dimensions (UCI KDD Archive 1999), is a well known labeled data that has been extensively used in evaluation of stream clustering experiments. Synthetic data sets of different sizes, with 6 numeric and 4 categorical dimensions, were generated using a customized data generator. Each of the synthertic data set consisted of six clusters in hyper-rectangular data space. The generated points were randomized and a time stamp was assigned to each point to simulate incremental data.

S.No.	Dataset	Size (GB)	Instances (10^6)
1	KDD-99 Cup	0.7	4.8
2	syn-8GB	8	160
3	syn-12GB	12	240
4	syn-16GB	16	320
5	syn-20GB	20	400

Table 2: Description of datasets; syn-n: Synthetic data set

Scalability of the platforms: We observed from Figure 2(a) that run time of ExCC-MR exceeds that of ExCC-P on all dataset. Nephele is advantaged by the pipelining support for communicating data between PACTs, while Hadoop is constrained to communicate data between jobs through HDFS. Massive disk I/O by Hadoop retards the overall execution speed of the application. With increasing data size and consequent disk I/O, the increase in timings is steeper for ExCC-MR than for ExCC-P, indicating better scalability of Nephele-PACT to Hadoop-MR.

Comparison of resource usage We studied the impact of degree of parallelism (DoP) on the performance of Hadoop-MR and Nephele-PACT using synthetic dataset syn-12GB by measuring averages of CPU utilization, CPU waiting time and network bandwidth. Each experiment was carried out three times to average out the influence of routine system activities on measurements.

CPU Usage: Figure 2(b) shows average CPU utilization (%) (solid lines) and average CPU waiting time (%) (dot lines) with DoP varying from 18 to 42. For lower values, ExCC-P shows higher CPU utilization than ExCC-MR. With increasing DoP, utilization falls for Nephele and in-

[1]http://www.stratosphere.eu/faq

creases for MR. After DoP=36 the graph shows a dip for both frameworks because of the data size. For DoP=42, excess availability of resources leads to lower CPU utilization for both frameworks. We conjecture that the point of dip will depend on the data size. On the other hand, lower CPU waiting time of ExCC-P compared to ExCC-MR is due to Hadoop's constraint of writing data on HDFS before shipping to subsequent jobs. Waiting time for ExCC-MR are consistently higher than ExCC-P because of the routine file writing by Hadoop after each Map and Reduce. PACT passes data through network channel and hence CPU waiting time is negligible.

Network bandwidth usage: It is observed from Figure 2(c) that Nephele-PACT consumes lower bandwidth as compared to Hadoop-MR. This is despite the fact that Nephele uses network channel to ship data between different contracts in the DAG. Hadoop utilizes excessive network for writing as well as reading data from HDFS. The reason for the dip in the network usage at DoP=36, is that with increased parallelism data locality improves and read/writes happen on the local compute node with increased frequency for both frameworks. Naturally, PACT benefits more from this phenomenon, as it was already performing minimal read/write on disk.

Incremental Processing: Observations for processing of incremental data in batches vindicated our earlier conclusion regarding resource usage. Figure 3 shows CPU utilization and wait time for ExCC-MR and ExCC-P, while processing syn-20GB data set in incremental mode (batches of sizes $12{+}4{+}4$). Arrows in the figure are marked to reveal the beginning and end of each batch. Arrows A through D are for ExCC-MR and arrows E through H are for ExCC-P. The graph clearly reveals the differences in the two platforms, while showing striking similarity to Figure 2 (b). Similar observations for network bandwidth were observed and hence have not been shown in interest of brevity.

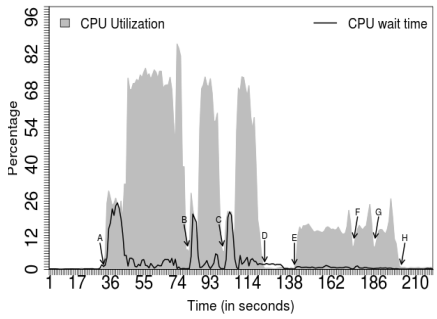


Figure 3: CPU utilization and waiting time for three batches of (12, 4, 4) GB for ExCC-MR (A - D) and ExCC-P (E - H);

5 Conclusion

Hadoop MapReduce and Nephele-PACT are two open source frameworks for data parallel computations, in addition to many other properitary solutions. In this paper, we document our experiences while designing and testing a scalable, incremental clustering algorithm for two frameworks. We describe the algorithm and its detailed implementation, highlighting the differences that arise because of the differences in two frameworks. As both frameworks operate on HDFS, experiments capture performance differences because of the frameworks.

We conclude that Nephele-PACT has better scalability than Hadoop-MR, due to use of multiple communication channels. Consequently, it makes better use of system resources. However, robustness of Hadoop-MR can not be ignore due to its excellent fault-tolerant nature. During the course of experiments, we never had to restart ExCC-MR, while ExCC-P required occasional intervention.

References

Agrawal, R.; Gehrke, J.; Gunopolos, D.; and Raghavan, P. 1998. Automatic Subspace Clustering of High Dimensional Data for Data Mining Application. In *Proceedings of the ACM SIGMOD*.

Alexandrov, A. et al. 2010. Massively Parallel Data Analysis with PACTs on Nephele. *Proceedings of the VLDB Endowment* 3(2):1625–1628.

Battré, D. et al. 2010. Nephele/PACTs: A Programming Model and Execution Framework for Web-Scale Analytical Processing. In *Proceedings of ACM Symposium on CC*, 119–130.

Bhatnagar, V.; Kaur, S.; and Chakravarthy, S. 2013. Clustering Data Streams using Grid-based Synopsis. *Knowledge and Information System*.

Bryant, R. E. 2011. Data-Intensive Scalable Computing for Scientific Applications. *Computing in Science and Engineering* 13(6):25–33.

Dean, J., and Ghemawat, S. 2004. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on OSDI*, 137–150.

Ghemawat, S.; Gobioff, H.; and Leung, S.-T. 2003. The google file system. In *Proceedings of the 19th ACM Symposium on OSP*, 29–43. ACM.

Pavlo, A.; Paulson, E.; Rasin, A.; Abadi, D. J.; DeWitt, D. J.; Madden, S.; and Stonebraker, M. 2009. A Comparison of Approaches to Large-scale Data Analysis. In *Proceedings of ACM SIGMOD*, 165–178.

Rajaraman, A., and Ullman, J. D., eds. 2012. *Mining of Massive Datasets*. Cambridge University Press.

Sakr, S.; Liu, A.; and Fayoumi, A. G. 2013. The Family of MapReduce and Large Scale Data Processing Systems. *CoRR* abs/1302.2966.

UCI KDD Archive. 1999. KDD CUP 99 Intrusion Data. http://kdd.ics.uci.edu//databases/kddcup99.

University of California, Berkeley. 2006. Ganglia: High Performance Monitoring Tool. http://ganglia.sourceforge.net.