INTERNATIONAL RESEARCH JOURNAL IN ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET) E - ISSN: 2454-4752 P - ISSN : 2454-4744
VOL 3 ISSUE 4 (2017) PAGES 2544 - 2548
RECEIVED : 05.07.2017 PUBLISHED : 03.08.2017
Aug 03, 2017

SCHEDULING DECISIONS IN STREAM PROCESSING FRAMEWORKS ON HADOOP CLUSTERS

A.Vignesh[1], K.Dhakshnamurthy[2], D.B.Shanmugam[3]

[1]M.Phil, Research Scholar, Dr.M.G.R.Chockalingam Arts College, Arni.

[2]Assistant Professor, Department of BCA, King Nandhivarman College of Arts & science, Thellar.

[3]Associate Professor, Department of MCA, Sri Balaji Chockalingam Engineering College , Arni.

ABSTRACT

Big data processing is a hot topic of today’s computer world. One of the key paradigms behind it is MapReduce—parallel and massively distributed model inspired by the map and reduce functions commonly used in functional programming. Due to its simplicity and general availability of standard implementations, the paradigm has been massively adopted on current computer clusters. Yet, MapReduce is not optimal for all big data problems. My work focuses on the area of an alternative paradigm—stream processing—which has multiple advantages over the MapReduce, e.g., it avoids persistent data storing if not required. The research aims at overcoming deficiencies of existing stream processing frameworks that prevent its wider adoption.

Basic scheduling decisions are discussed and demonstrated on naive scheduling of a sample application. The works presents a proposal of a novel scheduler for stream processing frameworks on heterogeneous clusters, which employs design-time knowledge as well as benchmarking techniques to achieve optimal resource-aware deployment of applications over the clusters and eventually better overall utilization of the cluster.

In particular, the work deals with scheduling problems of stream processing applications on heterogeneous clusters. Heterogeneity is a typical characteristic of today’s large data centers (caused by incremental upgrades and combinations of computing architectures, including specialized hardware such as GPU or FPGA) and advanced scheduling mechanisms can significantly increase efficiency of their utilization. The state-of-the-art research and development of stream processing and advanced methods of related scheduling techniques are discussed in this document. A special attention is paid to benchmark-based scheduling for distributed stream processing which also forms the core of my previous work and the proposed research towards my doctoral thesis. Finally, the concept of novel heterogeneity aware scheduler is presented first in the intuitive way and then discussed deeper on theoretical basis. The prototype of the scheduler is then described and promising results of basic experiments are showed.

1. INTRODUCTION

As the Internet grows bigger, the amount of data that can be gathered, stored, and processed constantly increases. Traditional approaches to processing of big data, e.g., the data of crawled documents, web request logs, etc., involves mainly batch processing techniques on very large shared clusters running in parallel across hundreds of commodity hardware nodes. For the static nature of such datasets, the batch processing appears to be a suitable technique, both in terms of data distribution and task scheduling, and distributed batch processing frameworks, e.g., the frameworks that implement the MapReduce programming paradigm, have proved to be very popular.

However, the traditional approaches developed for the processing of static datasets cannot provide low latency responses needed for continuous and real-time stream processing when new data is constantly arriving even as the old data is being processed. In the data stream model, some or all of the input data that are to be processed are not available in a static dataset, but rather arrive as one or more continuous data streams. Traditional distributed processing frameworks like MapReduce are not well suited to process data streams due to their batch orientation. The response times of those systems are typically greater than 30 seconds while real-time processing requires response times in the (sub) seconds range. To address distributed stream processing, several platforms for data or event stream processing systems have been proposed, e.g., S4 and Storm,. In this work, we build upon one of these distributed stream processing platforms, namely Storm. In the case of distributed batch processing, both resources allocation and tasks scheduling can be done prior to the processing of a batch of jobs based on knowledge of data and tasks for processing and of a distributed environment. Moreover, during batch processing, required resources are often simply allocated statically from the beginning to the end of the processing. In the case of distributed stream processing, which is typically continuous, dynamic nature of input data and unlimited processing time require dynamic allocation of shared resources and real-time scheduling of tasks based on actual intensity of input data flow, actual quality of the data, and actual workload of a distributed environment. For example, resource allocation and task scheduling in Storm involves real-time decision making considering how to replicate bolts and spread them across nodes of a cluster to achieve required scalability and fault tolerance.

2. RELATED WORK

Cluster analysis is an exploratory data analysis tool where there are no pre-set classes, although the number of classes may be set. Because in cluster analysis classes must be constructed without guidance it is known as an unsupervised learning technique. This is akin to how people or animals learn about their environment when they are not told or directed what to learn. Clusters are formed when attributes of observations tend to vary together. Cluster analysis constructs "good" clusters when the members of a cluster have a high degree of similarity to each other (internal homogeneity) and are not like members of other clusters (external homogeneity). However, there is no agreement over how many clusters a dataset should be partitioned into. There are no guidelines on the number of clusters that would be optimal to aid supervised learning efforts. Statisticians have developed clustering procedures which group observations by taking into account various metrics to optimize similarity. The major type of cluster analysis, which will be used in this study, is hierarchical clustering.

Hierarchical clustering begins with putting each observation into a separate cluster. Clusters are then combined successively based upon their resemblance to other clusters. The number of clusters is reduced until only one cluster remains. A tree or dendrogram can represent hierarchical clustering. Each fork in the tree represents a step in the clustering process. The tree can be sectioned at any level to yield a partition of the set of observations. At its early stages the dendrogram is very broad. There are many clusters that contain very similar observations. As the tree structure narrows the clusters comprise coarser, more inclusive groupings.

In my banking study the same thing happened. The bank was surprised to find it had different types of account holders, some of which were not profitable. They were then able to focus on the profitable ones, and either disengage or convert the non-profitable ones. In essence, developing different sets of rules for cluster subgroups. I think this works for almost any types of dataset. …In data mining contexts, it probably works best with large datasets, because there's always the hope that you might get a surprise hidden in a lot of data (e.g. the profitability of account types) or discover a nugget of hidden data (e.g. in the context of health insurance claims, a tiny group of fraudsters operating a scam) (Wishart, 1999).

A further example Wishart mentions comes from the field of astronomy. In the Hertzsprung-Russel diagram stars are plotted by temperature and luminosity. "Dwarf" and "giant" stars are in separate clusters. Within each cluster there is a different relationship between temperature and luminosity. The correlation is negative for the dwarfs and positive for the giants. If just one correlation were figured for the dataset of all stars the correlations within the two clusters would wash each other out. This would erroneously indicate no relationship between temperature and luminosity. Yet within the clusters for the two types of stars there are clear "rules" governing the relationship between temperature and luminosity.

3. EVOLUTION MEASURES OF HETROGENEOUS STREAMING

Memory is managed across domains using an abstraction called buffers, which are used to help manage properties and track dependences. These three component building blocks offer abstractions that enhance programmer productivity, provide transparency and control, and enable a separation of concerns between the scientist programmer and the one tuning for a target architecture. A domain is a set of computing and storage resources which share coherent memory and have some degree of locality. Examples of domains include a host CPU, a Knights family co-processor card, a node in a cluster reached across the fabric, a GPU, and a subset of cores that share a memory controller.

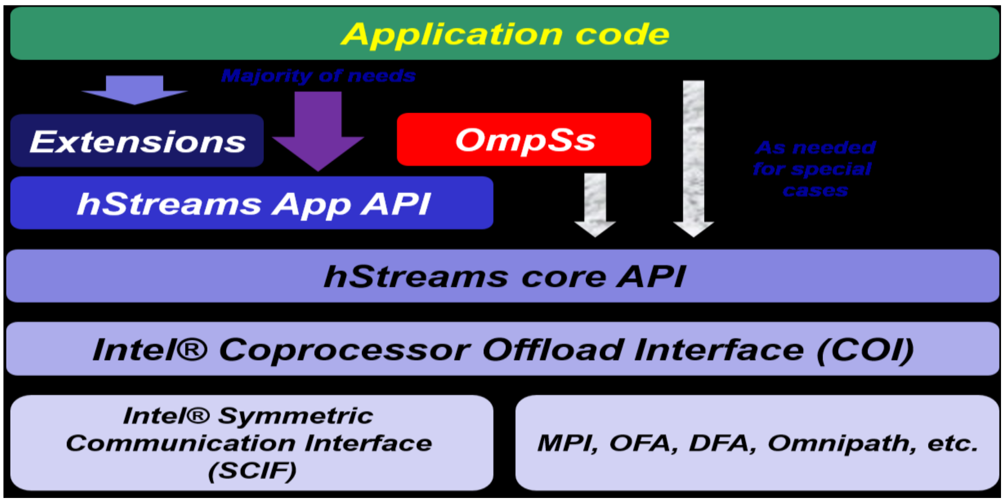
Application code
Majority of needs
Extensions
OmpSs
As needed for special cases
hStreams App API
hStreams core API
Intel® Coprocessor Offload Interface (COI)
Intel® Symmetric Communication Interface (SCIF)
MPI, OFA, DFA, Omnipath, etc.

Fig.1. HStreams APIs layered between fabric interfaces & higher abstractions

They are free to execute and complete out of order, as long as the effect of such optimizations is not visible at the semantic level, i.e., they do not violate the sequential FIFO semantic of the stream. Tasks naturally expand across a stream's threads when they use threading constructs like OpenMP or

TBB. Buffers encapsulate memory. They are used to manage storage properties (e.g. memory type and affinity) and track dependences among actions. All memory that can be referenced by user code is represented in a unified source proxy address space, which is partitioned into buffers. The virtual

4. RESULT ANALYSIS

Decomposition increases concurrency. Tiling can help in that a smaller amount of data needs to get transferred to a computing resource before the work can begin on it. Some or all of the communication latency can get covered by computation, through pipelining. Decomposition creates a larger number of tasks, which may be more evenly divisible by the number of computing resources, leading to less load imbalance. The best degree of tiling and number of streams depends on the matrix size and algorithm. Users want to be able to tune these easily, by changing just a few parameters. We provide a detailed analysis of that technique in for matrix multiply, Cholesky, and LU, but only for offloading to a single KNC card rather than multiple MICs or MICs plus a host.

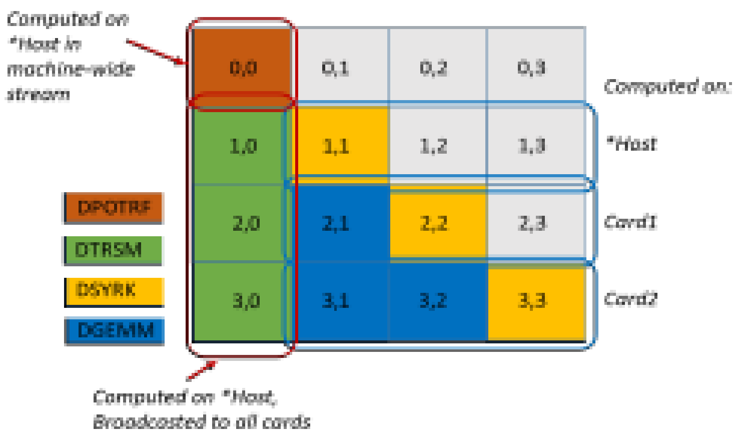

Fig.2. Decomposition of the matrix into tiles and distribution

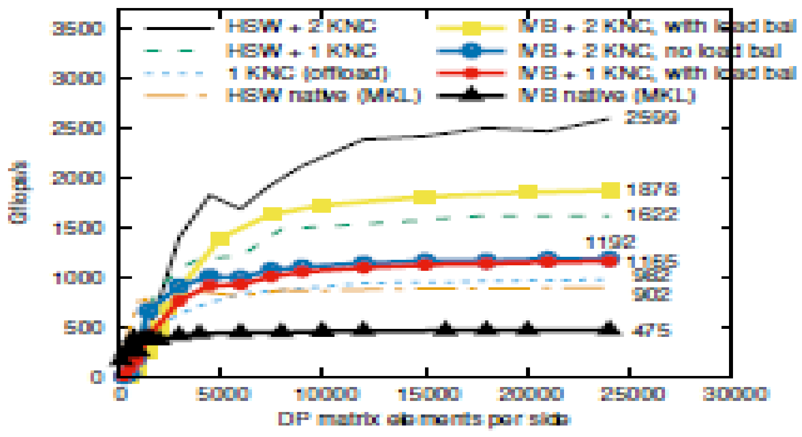

Fig.3.Output Analysis

CONCLUSION

This thesis briefs about the stream data mining, its need and the challenges associated in mining potentially infinite data streams along with various stream mining algorithms for classification and clustering. The four dimensions of streaming data mining discussed in the thesis covers the study of this field completely and in modular way. It specifies the need of new algorithms and evaluation measures relevant to this field and mentioned some of them used in stream mining scenario. The various available tools or platforms to provide the appropriate framework to deal with large scale data streams along with their key features have also been described in chronological order that helped in undertaking the evolvement of the streaming data computing and mining platforms.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters,"

Communications of The ACM, vol. 51, no. 1, pp. 107--113, 2008.

[2] R. Lämmel, "Googles MapReduce programming model - Revisited," Science of Computer Programming, vol. 70, no. 1, pp. 1--30, 2008.

[3] M. Elteir, H. Lin and W.-c. Feng, "Enhancing MapReduce via Asynchronous Data Processing," 2010.

[4] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy and R. Sears, "MapReduce Online," 2010.

[5] E. Mazur, B. Li, Y. Diao and P. Shenoy, "Towards Scalable One-Pass Analytics Using MapReduce," pp. 1102--1111, 2011.

[6] B. Lohrmann, D. Warneke and O. Kao, "Nephele streaming: stream processing under QoS constraints at scale," Cluster Computing, pp. 1-18, 2013.