E500 Introduction to ISE
Report

Coarse-grained and Fine-grained Analytics in a Single Framework

Dilshan Niranda Perera (dnperera@iu.edu)

05 November 2018

The report below is inspired by the Global AI Supercomputer concept presented by Prof. Geoffrey Fox and Dr. Judy Qiu. It is closely related to the Twister2 computing framework [1] developed at the Digital Science Center of the Indiana University.

1 Introduction

The amount of data generated by day-to-day operations have been exponentially increasing during the past couple of decades. The problem of analyzing and decision making based on these data, has also convoluted owing to the increased data volumes, diversified/ unstructured formats and added inconsistencies in the data streams. Advent of the World Wide Web, the Internet, Internet of Things (IoT), etc. are some technologies which have directly contributed to this paradigm shift. Hence the term, "Big Data" has now become a norm in the scientific domains. In addition to the accumulation of terabytes of data, consumers/ users of the insights of data have also grown in numbers, they come up with more complex requirements in the form of queries, and the demand for faster insights is ever increasing. Hence, there is an urgent need for a data analytics framework to ingest and analyze such data workloads.

Furthermore, parallel and distributed computing are essential to process big data owing to the data being naturally distributed and processing often requiring high performance in compute, communicate and I/O arenas. Over the years, the High Performance Computing community has developed frameworks such as message passing interface (MPI) [2] to execute computationally intensive parallel applications efficiently. HPC applications target high performance hardware, including low latency networks due to the scale of the applications and the required tight synchronous parallel operations. Big data applications have been developed for commodity hardware with Ethernet connections seen

in the cloud. Because of this, they are more suitable for executing asynchronous parallel applications with high computation to communication ratios. Recently, we have observed that more capable hardware comparable to HPC clusters is being added to modern clouds due to increasing demand for cloud applications in deep learning and machine learning. These trends suggest that HPC and cloud are merging, and we need frameworks that combine the capabilities of both big data and HPC frameworks.

A common concern in big data frameworks is that the data can be too big to fit into the memory of a node or a cluster. In another aspect, the load balancing of data, as the data is generated across an extreme variety of data sources. Temporal variation of arrival times of data is another concern, which requires the frameworks to add batch data processing as well as streaming data processing in the pipeline. Difference sets of data may have variable processing time across data and iterations of algorithms.

Another major concern in the big data analytics arena is the question of scheduling workloads, whether to use coarse-grained scheduling or fine-grained scheduling. Coarse-grained scheduling would allow the framework to spawn executing elements at the start of the framework, and schedule workloads dynamically among these computing elements. While, fine grained scheduling framework would spawn executing elements as and when it receives a workload. Coarse-grained scheduling is deemed as a more favorable apporach for realtime/ streaming data analytics, while fine-grained has an inherent overhead of acquiring resources before execution. Apache Spark [3] is a very popular framework which utilizes coarse-grained resource scheduling. While Spark has been successful in data analytics using the resilient distributed datasets (RDDs) [4], there is a perception in the high performance computing community that people often use Spark as a framework for distributed computing/ workload distribution rather than using it for data analytics. Due to this, the users are limited to use coarse grained scheduling, where as for the purpose of distributed computing and workload distribution, a combination of coarse grained and fine-grained scheduling together would be much more beneficial.

This article attempts to propose a model in which both the coarse-grained and fine-grained resources can scheduled depending on the users requirements. Digital Science Center at the Indiana University is currently involved in elevating its data analytics platform, *Twister2* to have this capability, and and the author sees it as an interesting research topic which he would be personally involved in. The rest of the article is organized is as follows. Section 2 would explain the dataflow model, section 3 would look at a brief history of dataflow and data analytics frameworks. Section 4 will discuss the *Twister2* framework. Section 5 will discuss the new additions proposed to add the aforementioned capabilities.

2 Dataflow Model

The dataflow computation model has been presented as a way to hide some of the system-level details from the user in developing parallel applications. It is a software paradigm based on the idea of disconnecting computational actors into stages (pipelines) that can execute concurrently. Dataflow can also be called stream processing or reactive programming. The most obvious example of data-flow programming is the subset known as reactive programming with spreadsheets. As a user enters new values, they are instantly transmitted to the next logical "actor" or formula for calculation. With dataflow, an application is represented as a graph, more specifically a DAG (Directed Acyclic Graph) with nodes doing computations and edges indicating communications between the nodes. A computation at a node is activated when it receives events through its inputs. A well-designed dataflow framework hides the low-level details such as communications, concurrency, and disk I/O, allowing the developer to focus on the application itself.

Every major big data processing system has been developed according to the dataflow model, and the HPC community has also developed asynchronous many tasks systems (AMT) according to the same model. AMT systems mostly focus on computationally intensive applications, and there is ongoing research to make them more efficient and productive. We find that big data systems developed according to a dataflow model are inefficient in computationally intensive applications with tightly synchronized parallel operations, while AMT systems are not optimized for data processing.

At the core of the dataflow model is an event-driven architecture where tasks act upon incoming events (messages) and produce output events. In general, a task can be viewed as a function activated by an event. The cloud-based services architecture is moving to an increasingly event-driven model for composing services in the form of Function as a Service (FaaS). FaaS is especially appealing to IoT applications where the data is eventbased in its natural form. Coupled with microservices and server-less computing, FaaS is driving next-generation services in the cloud and can be extended to the edge.

3 History

Data workloads have been conventionally categorized into batch and streaming. Recently emergence of machine learning has added another dimension to the data analytics workloads. During the past couple of decades, there were several 'waves' of analytics frameworks developed to support these workloads. Advent of Massage Passing Interface (MPI) [5], Bulk Synchronous Parallel (BSP) model [6] were some early notable technological breakthroughs. Followed by BSP, it was the era of Map-Reduce model [7], one of the most successful Big Data

technology paradigms of the recent years. There were number of technology frameworks built upon Map Reduce, such as Apache Hadoop [8], Apache Hive [9], Apache Pig [10]. These technologies were catering to solve the batch analytics problem.

Recently, Apache Spark [11] has taken over Hadoop to be the dominant technology in the batch analytics domain. Streaming analytics came into the Big Data domain with the popularization of IoT devices and the internet, which generates large number of sparse, ad-hoc, diverse data streams. Apache Spark Streaming [3], Apache Storm [12], Apache Flink [13, 14], Apache Samza [15] are the dominant streaming analytics frameworks at present. Machine learning (ML) has opened up a new dimension in the Big Data analytics domain, where users can develop models on top of large volumes of data. Apache Spark Mllib [3] and Apache Mahout [16] are the pioneering frameworks in the machine learning domain and Harp-DAAL Project [17] is an effort to introduce ML capabilities on Hadoop.

History of data analytics frameworks mentioned here, can be categorized into mainly 4 epochs.

- Generation 1:
  Map-Reduce (MR) based data analytics frameworks merely focusing on Batch data analytics. This was the inception of data analytics led by Google. The framework only had a crude set of tools to crunch a considerable set of data which could not have been efficiently analyzed by the traditional RDBMS (Relational DataBase Management) systems.

- Generation 2:
  Second generation of data analytics frameworks introduced interactive analytics into the picture. Apache Tez is a good example of this. It is an extensible framework for building high performance batch and interactive data processing applications, coordinated by YARN in Apache Hadoop. Second generation improved the MapReduce paradigm by dramatically improving its speed, while maintaining MapReduce's ability to scale to petabytes of data. A common characteristic of these frameworks was that they were using dataflow concepts as a Directed Acyclic Graph (DAGs)

- Generation 3:
  Third generation of data analytics frameworks introduced realtime data analytics and iterative processing into the frameworks. Apache Spark was the most notable of these frameworks for their contribution of using Resilient Distributed Datasets (RDDs) for data analytics. These frameworks are the most commonly used data analytics tools even in the current industry. Because of their ability to handle near-real time streaming workloads.

- Generation 4:
  Apache Flink claims that they are a ground breaking framework where they introduce true real time streaming analytics in to the industry. These fraworks are capable of handling hybrid dataflows which contained batch and streaming analytics. These also attempt to support cyclic dataflows and a native iterative processing approach.

4 Workload Scheduling

Granularity is the extent to which a system is broken down into small parts, either the system itself or its description or observation. It is the extent to which a larger entity is subdivided. For example, a yard broken into inches has finer granularity than a yard broken into feet. As mentioned in the Introduction, there are two main schools of thought when it comes to workload scheduling granularity. First is, *Coarse-grained* scheduling and the other is *Fine-grained* scheduling.

4.1 Fine-grained scheduling

In "fine-grained" mode, each executor runs as a separate computing task. This allows multiple instances of workloads to share cores at a very fine granularity, where each application gets more or fewer cores as it ramps up and down, but it comes with an additional overhead in launching each task. This mode may be inappropriate for low-latency requirements like interactive queries or serving web requests.

In a JVM (Java Virtual Machine) based environment, eventhough tasks in fine-grained will relinquish cores as they terminate, they will not relinquish memory, as the JVM does not give memory back to the Operating System. Since most of the data analytics frameworks were Java based, this was seen as a major bottleneck. Additionally, executors are brought up lazily, hence it would add more overhead to the task execution.

4.2 Coarse-grained scheduling

In "coarse-grained" mode, each executor runs as a single resource task. Executors are brought up eagerly when the application starts, until a specified maximum is reached. The application then has the ability to utilize these resources for execution.

The benefit of coarse-grained mode is much lower startup overhead, but at the cost of reserving computing resources for the complete duration of the application. To alleviate this bottleneck, frameworks have attempted to use dynamic resource allocation, which could be seen as a hybrid approach.

The downside of resorting into either of these extremes would constrain data analytics platforms to use a below par resource scheduling mechanism for their usecases. As we see, the resource scheduling is very much dependant on the application and the users should not be constrained to use a particular resource because of the limitations of the framework.

The current issue with the granularity of the workloads is, users tend to select frameworks such as Apache Spark purely on the basis of distributing their workloads, because of its convenient API. This is not the intended use of such frameworks and hence it would be suboptimal in the long run.

4.3 Unified Batch, Streaming and Iterative Analytics

It is an evident fact in the Big Data analytics community that a practical application data analytics scenario encompasses more than one out of the batch, streaming and iterative data analytics approaches. But, there is no unified framework which could support this requirement. Murry et al [18] identified this opportunity when they developed the framework Naiad. More recently, Google has re-looked at the analytics problem with a different point of view, when they introduced the Google Cloud Dataflow engine [19], where they look at streaming analytics as unbounded data streams and batch analytics as bounded data sets. They propose a hybrid approach, where they combine both types of workloads using a common programming environment, Apache Beam [20].

5 Twister2

Twister2, developed by the Digital Science Center of the Indiana University, provides a data analytics hosting environment where it supports different data analytics including streaming, data pipelines and iterative computations. Unlike many other big data systems that are designed around user APIs, Twister2 is built from bottom up to support different APIs and workloads. Our vision for Twister2 is a complete computing environment for data analytics. One major goal of Twister2 is to provide independent components, that can be used by other big data systems and evolve separately.

The current implementation of Twister2 is capable of handling batch, streaming and iterative data analytics workloads independently. The goal of the author's study is to merge the capabilities into a unified environment. The study encompass the area of enabling *Twister2* to handle coarse grained and fine grained workloads on the same environment.

6 Hierarchical Task Graphs (HTG)

The main point of extension for the new capability is hierarchical task graph (HTG). This would look at coarse-grained workloads and fine-grained workloads as a hierarchy of data flow graphs and schedule the resources accordingly. Hence, this would allow users to have the autonomy of resources like in the fine-grained resource scheduling without compromising the benefits of coarse grained resource scheduling.

The hierarchical task graph strategy is to compose multiple independent/dependent dataflow task graphs into a single entity. A dataflow task graph consists of multiple subtasks which are arranged based on the parent-child relationship between the tasks. In general, a dataflow task graph consists of multiple vertices and edges to connect those vertices. The vertices represent the characteristics of computations and edges represent the communication between those computations. The simple representation of hierarchical task graph is shown in Figure 1.

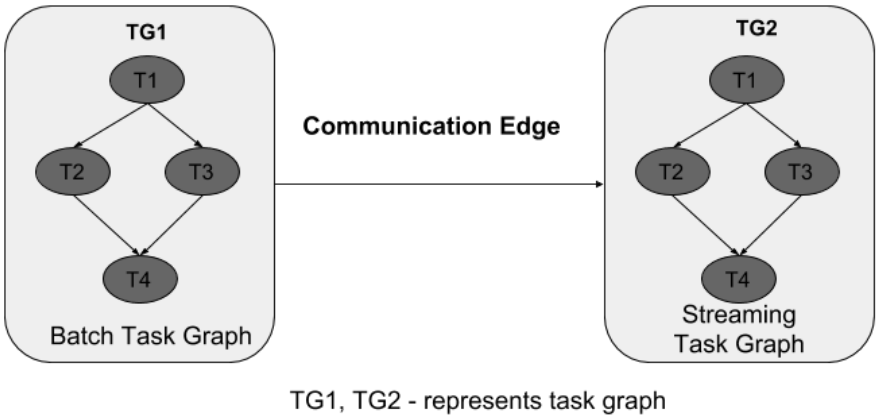

Figure 1: Example Hierarchical Task Graph

6.1 Related Work

A similar approach was proposed by Zhe Ma et al [21], where they propose optimal and fast heuristic algorithms to schedule task clusters based on interleaving subtasks. Their algorithm will take into account the situation when multiple tasks are required to run concurrently and interleave their separated subtasks schedules to generate a new united schedule. Even though it is not directly related, it is one of the first instances where scheduling is done in a hierarchical manner.

Alefragis et al [22] proposed a method that uses a MIP model to solve individual DAG sub-problems to optimality and a heuristic approach to solve the whole problem using a bottom up traversal of the HTG tree. Xia et al [23], also provided another aspect of HTG, where they bring a hierarchical scheduling method which schedules DAG structured computations at three different levels on manycore systems. This is interesting because, our approach is more similar to this, eventhough this scheduler is for a static set of resources.

Modern datacenter schedulers is also an interesting area to look into. Delgado et al [24] presented 'Hawk', a hybrid scheduler, which falls in a middle ground between centralized and distributed schedulers where it centralizes the scheduling of long jobs and schedules the short jobs in a distributed fashion. 'Sparrow' is another scheduler by Ousterhout et al [25], which takes a purely distributed resource scheduling approach. 'Mercury' is another interesting approach by Karanasos et al [26] where they propose a hybrid resource management architecture. The key insight is to offload work from the centralized scheduler by augmenting the resource management framework to include an auxiliary set of schedulers that make fast/distributed decisions.

6.2 Methadology

The main objective of the HTG task scheduler approach is to overcome the issues and limitations in the existing centralized and distributed task scheduler for processing the big data analytics applications. The centralized task scheduler creates huge traffic because all the requests have to go forward and backward from the workers to the central task scheduler that leads to a performance bottleneck. It provides poor scheduling decisions for long running jobs. However, the distributed task scheduler has the opportunity to make the scheduling decisions in a distributed way.

However, the combination of centralized and the distributed task scheduler (hybrid task scheduler) achieves high performance by improving the throughput of the submitted tasks as well as it reduces the traffic. The centralized task scheduler acts as a controller which identify the task graphs available in the hierarchical task graph and schedule the task graphs to the workers. The distributed task scheduler which is running in every worker makes the scheduling decisions to run their individual tasks. The initial design of hybrid task scheduler is shown in Figure 2.

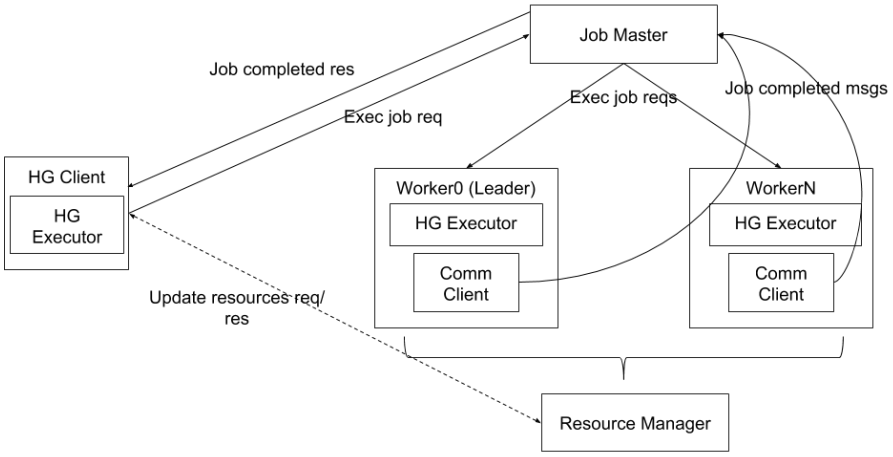

Figure 2: Example Hierarchical Task Graph

The idea is for the users to create a Hierarchical Task Graph consists of both streaming and batch jobs and pass the jobs using a client (HTG Client). HTG Client will be responsible of assigning resources for each tasks in the HTG meta graph. In each task, users will have the full control over the resources it receives and steps such as resource allocation etc, would be transparent to the task. Once resources are allocated to a particular task, a message will be sent from the HG Client to the workers, specifying which task to process. Upon completion of tasks, workers will send a message to the client that they have completed the work. These communications are routed via the job master process. If the downstream tasks have a data dependency, it will be transferred using the dataset API of the framework. Depending the resource requirement of the downstream tasks, client will update the resources assigned to the job and send another message to executed the new tasks. This process will continue until the client completes all the tasks designated in the HTG graph.

7 Discussion

Since the advent of Map-Reduce paradigm, data analytics have come a long way in computer engineering domain. There have been number of data analytics frameworks in the industry at the moment. Resource allocation and scheduling is at the core of all of these data analytics frameworks. Coarse grained and fine grained resource scheduling are the two primary approaches these frameworks have adopted. The current issue with the granularity of the workloads is, users tend to select frameworks purely on the basis of distributing their workloads, because of its convenient API. This is not the intended use of such frameworks and hence it would be suboptimal in the long run. Hence, from this report,

the authors are trying to propose a new scheme of resource scheduling called Hierarchical Task Graphs, where both coarse grained and fine grained resource scheduling can be allocated in the same environment. This would be a more transparent and composable way to deal with resource scheduling and it would give more flexibility to the users to make better use of the resources in streaming, batch as well as iterative workloads.

References

[1] Twsiter2. [Online]. Available: https://twister2.gitbook.io/twister2

[2] Open mpi - a high performance message passing library. [Online]. Available: https://www.open-mpi.org/

[3] Apache spark. [Online]. Available: https://spark.apache.org/

[4] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[5] D. W. Walker, "The design of a standard message passing interface for distributed memory concurrent computers," *Parallel Computing*, vol. 20, no. 4, pp. 657–673, 1994.

[6] A. V. Gerbessiotis and L. G. Valiant, "Direct bulk-synchronous parallel algorithms," *Journal of parallel and distributed computing*, vol. 22, no. 2, pp. 251–267, 1994.

[7] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[8] Apache hadoop. [Online]. Available: https://hadoop.apache.org/

[9] Apache hive. [Online]. Available: https://hive.apache.org/

[10] Apache pig. [Online]. Available: https://pig.apache.org/

[11] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[12] Apache storm. [Online]. Available: https://storm.apache.org/

[13] Apache flink. [Online]. Available: https://flink.apache.org/

[14] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[15] Apache samza. [Online]. Available: https://samza.apache.org/

[16] Apache mahout. [Online]. Available: https://mahout.apache.org/

[17] Harp-daal framework - indiana university. [Online]. Available: https://dsc-spidal.github.io/harp/

[18] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, "Naiad: a timely dataflow system," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 439–455.

[19] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt *et al.*, "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[20] Apache beam. [Online]. Available: https://beam.apache.org/

[21] Z. Ma, F. Catthoor, and J. Vounckx, "Hierarchical task scheduler for interleaving subtasks on heterogeneous multiprocessor platforms," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*. ACM, 2005, pp. 952–955.

[22] P. Alefragis, C. Gogos, C. Valouxis, G. Goulas, N. Voros, and E. Housos, "Assigning and scheduling hierarchical task graphs to heterogeneous resources."

[23] Y. Xia, V. K. Prasanna, and J. Li, "Hierarchical scheduling of dag structured computations on manycore processors with dynamic thread grouping," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2010, pp. 154–174.

[24] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel, "Hawk: Hybrid datacenter scheduling," in *Proceedings of the 2015 USENIX Annual Technical Conference*, no. CONF. USENIX Association, 2015.

[25] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 69–84.

[26] K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G. M. Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga, "Mercury: Hybrid centralized and distributed scheduling in large shared clusters." in *USENIX Annual Technical Conference*, 2015, pp. 485–497.