Task Scheduling in Big Data - Review, Research Challenges, and Prospects

Kannan Govindarajan*, Supun Kamburugamuve†, Pulasthi Wickramasinghe‡, Vibhatha Abeykoon§ and Geoffrey Fox¶
School of Informatics and Computing, Indiana University
Bloomington, USA
Email:*kannan.gridlab@gmail.com, †supun06@gmail.com, ‡pulasthi911@gmail.com, §vibhatha@gmail.com, ¶gcfexchange@gmail.com

***Abstract*—In a Big data computing, the processing of data requires a large amount of CPU cycles and network bandwidth and disk I/O. Dataflow is a programming model for processing Big data which consists of tasks organized in a graph structure. Scheduling these tasks is one of the key active research areas which mainly aims to place the tasks on available resources. It is essential to effectively schedule the tasks, in a manner that minimizes task completion time and increases utilization of resources. In recent years, researchers have discussed and presented different task scheduling algorithms. In this research study, we have investigated the state-of-art of various task scheduling algorithms, scheduling considerations for batch and streaming processing, and task scheduling algorithms in the wellknown open-source big data platforms. Furthermore, this study proposes a new task scheduling system to alleviate the problems persists in the existing task scheduling for big data.**

***Keywords—Big Data, MapReduce, Dataflow, Task Scheduling Model, Twister2, Static and Dynamic Task Scheduling*.**

I. INTRODUCTION

In recent days, many applications generate Big data such as Facebook, Google, general public websites, scientific experiments, commercial applications, cloud applications, IoT devices, e-governance applications, bio-medical applications, and much more. Big data [1], [2] represents the characteristics of volume, veracity, and variety of data. It also [3] represents the data collected in a systematic manner but exceeds the storage and power capacity of typical machines in an organization. The rapid growth of data volume which requires processing petabytes of data per day. It describes the exponential growth and availability of structured, semi-structured and unstructured data. Furthermore, the data consists of audio, video, images, and more. It should be processed properly in order to make accurate and timely decisions. Hence, it is essential to process and extract such data to understand the meaningful insights hidden within such data, this is known as Big data analytics.

Big data analytics is empowered by machine learning or statistical algorithms to process big data and understand meaningful information out of it. Consequently, it is important to process those data within a limited time. Major big data frameworks have been developed according the dataflow model which represents a computation as a graph consisting of processing nodes and communicating edges. The Big data platforms such as Hadoop [4], Spark [5], Flink [6], Heron [7] and others are examples of such systems. These systems are required to execute jobs that take less than a minute to many hours to long running as in streaming jobs. The performance of Big data platforms depends on how effectively both the workloads are handled and processed in an efficient way. In general, big data applications or jobs consists of the asynchronous tasks in the form of functions. For example, map and reduce are well-known functions in big data systems. The map task processes each input data block and produce the intermediary results whereas the reduce tasks process the intermediary results and produce the final output. Additionally, big data applications may receive input data in a batch mode or streaming mode. Hence, it is mandatory to effectively schedule those tasks with an appropriate input data.

The process of scheduling the tasks into the cluster resources in a manner that minimizes task completion time and resource utilization is known as Task Scheduling. The main functional requirements of task scheduling are scalability, dynamism, time and cost efficiency, handling different types of processing models, data and jobs, and so on. The other major objectives of task scheduling are to schedule the independent and dependent tasks and reduce the number of task migrations in an optimal way which improves the computation time and utilization of cluster resources. In addition to that, the big data platforms construct the task graph per job which can be generated either in a static or dynamic mode. Hence, the task scheduling should have the ability to handle and schedule both static and dynamic task graphs.

A dataflow framework designed to process data, consists of well defined layers such as communication, resource scheduling, task system, and distributed data abstraction. The design decisions made at each layer determines the the applications supported efficiently. We find that modern systems are designed with a fixed set of design choices at these layers rendering them suitable for narrow set of applications. Twister2 [8] is a big data system designed to overcome some of the shortcomings of monolithic designs of current big data systems by introducing a clear component based approach to big data. Because of this, Twister2 needs to support a broad range of task scheduling capabilities. We take Twister2 as an integral part of our discussion to introduce some of the

978-1-5386-4349-5/17/$31.00 ©2017 IEEE

requirements of big data application. The main contributions of this research paper are summarized below:

- Investigated various static and dynamic task scheduling algorithms and task scheduling considerations for processing of batch and streaming jobs.
- Explored the task scheduling algorithms available in the popular open-source big data platforms such as Hadoop, Mesos, Spark, Flink, Heron, and Storm for batch and stream job processing.
- Proposed a task scheduling model/system which considers both static and dynamic task graphs and provides the ability to schedule batch, streaming, MPI, and micro-services job types.

The paper is organized as follows: Section II introduces the task based systems. Section III investigates the classification of various static and dynamic task scheduling algorithms. Section IV discusses the various scheduling considerations to be considered for batch and streaming task scheduling. Section V explores the task scheduling systems in various popular big data tools or platforms. Section VI presents the overview of Twister2 and proposed task scheduling model. The summary of findings and future research directions are discussed in section VII. Section VIII concludes this research paper followed by the future work.

II. TASK BASED SYSTEMS

The dataflow programming model [9] is mainly designed to simplify the processing of large-scale data. The dataflow model hides the underlying details of distributed processing, coordination, and data management. It also simplifies the process of specifying the task parallelism and dynamically determining the dependency between the tasks. According to dataflow model, an application is defined as a graph where vertexes represent task computations and edges represent communications between those tasks. The graph defined by the user is termed user graph and this is turned into a graph that can be executed on the available resources. The graph running on the physical machines is called the execution graph or physical graph. The user graph can be generated both dynamically and statically. The allocation of the graph nodes to the resources is handled by the task scheduler. Depending on the information available and how the graph is generated the task scheduler can do static schedules, dynamic schedules including task migrations. The most popular big data systems are designed based on the dataflow model principle and there are some High Performance Systems designed according to the same model with tasks.

Javier Conejero et al. [10] proposed a task-based programming framework known as COMPS. It is designed to facilitate the development of applications for distributed computing infrastructure. It is a worthwhile alternative for task-based programming model for big data applications. It achieves scalability and elasticity through cloud virtual machines. Their runtime system is capable to identify the implicit parallelism of the applications during the execution time, which enables the execution of an application in a distributed infrastructure. It supports various functionalities such as data dependency analysis, task scheduling, and fault tolerance. Task scheduling is responsible for allocating the tasks to resources which considers the various constraints such as data-locality constraints, task constraints (soft hard), and resource workload constraints. The task scheduling receives data locality and replica information from the data info provider. Fredy Juarez et al. [11] proposed a task scheduler for COMPS which considers the data locality, task constraints, and the workload of the resource for assigning the task to the distributed resources. Their proposed task scheduler is designed with an objective of minimizing the consumption of energy.

Michael Bauer et al. [12] designs a data centric parallel programming system along with deferred execution framework for heterogeneous applications. In this system tasks are created in a form of a tree structure and and each task can create sub-tasks providing asynchronous task execution. This model provides a dynamic task registration and sub-task registration to tear up larger tasks into smaller tasks. In Legion, the tasks are being launched by a launcher object which is launched automatically by the runtime but these launcher objects are executed by the task launchers. Sean Treichler et al. [13] introduces a system called Realm which is an event based runtime which allows distributed memory machines along with non-blocking runtime actions also provides an idea of task management and execution.

III. CLASSIFICATION OF TASK SCHEDULING ALGORITHMS

The task scheduling algorithms are mainly classified into two types namely static and dynamic task scheduling algorithms.

A. Static Task Scheduling and Algorithms

The static task scheduling allocates the tasks to the compute nodes before starting the execution of a task and also it knows the details about the compute nodes during compile time itself. It allows the execution of tasks continuously without any interruption until the completion of the task. It mainly reduces the scheduling complexity that generally happens during the runtime of the execution and reduces the number of compute nodes. But, the main drawback of the static task scheduling is failing to consider the workload of the resources and job-resource requirements that obviously leads to overutilization or under-utilization of the cluster resources which creates a situation for job execution failure. Some of the task scheduling algorithms are Capacity Scheduling, Data LocalityAware Scheduling, Round Robin Scheduling, and so on. It is impossible to discuss all the static task scheduling related research papers within this paper. Hence, we briefly discussed some of the closely related static task scheduling works in this section.

Ghodsi et al. [14] proposed the fair scheduling which aims to address the fair allocation (achieving statistical multiplexing) of resources by dividing the available resources using the max-min fair sharing. The fair scheduler allocates the available resources based on the memory by default but, it can be configured to schedule based on the CPU and memory values. Capacity Scheduler [15] is a pluggable scheduler to

Hadoop which is designed to execute multiple jobs concurrently by empowering with multiple queues/pools. Each queue/pool is guaranteed to allocate some fraction of cluster resources. The capacity scheduler supports the features such as hierarchical queues, guaranteed capacity, security, elasticity, and multitenant. Yintian Wang et al. [16] proposed the Round Robin scheduling algorithm which allocates the computing resources in a time-sliced manner, however, in the big data computing it allocates the computing slots to the tasks in a round-robin mode. Their proposed scheduling mechanism is implemented with multi-level feedback approach for reducing the response time of the big data applications. Jiang Bo et al. [17] proposed a data locality-aware Scheduler, data locality is the measurement of data localization of input data and performs the task scheduling based on the availability of input data in the cluster resources.

Yu-Chon Kao and Ya-Shu Chen [18] proposed a data locality-aware MapReduce scheduling framework for achieving the guaranteed quality of service to the interactive MapReduce applications. Their proposed scheduling mainly aimed to address two scheduling issues namely (i) scheduling a job with multiple map and reduce tasks (achieving end-toend deadline) and (ii) partitioning tasks to data-locality aware cluster resources (maximizing schedulable tasks). First Fit Decreasing Packing [19] is a heuristic bin-backing scheduling technique which is designed with an objective of accommodating m number of different task objects into n number of finite resources which reduces the number of resources to be used for the execution of a job. Chen He et al. [20] proposed a matchmaking scheduling technique which aimed to improve the data locality by avoiding unnecessary data transmissions. It doesnt require the delay factor D. The core idea of their scheduling technique is giving more preference to local map tasks than non-local map tasks. A locality marker is included to mark the nodes which ensure that each node gets their local tasks. It gives the relaxation for the strict job order for assigning tasks and achieves better performance than delay scheduling technique.

B. Dynamic Task Scheduling and Algorithms

The dynamic task scheduling takes the scheduling decisions during the runtime of task execution. It mainly considers the resource requirement, availability of resources, interprocess and inter-node traffic, energy efficiency, and more. It provides the support for migrating the task based on the availability of cluster nodes and workload of an application. Resource-aware scheduling, energy-aware scheduling, and deadline-aware scheduling are some of the most popular dynamic task scheduling algorithms. It is mainly aimed to utilize the resources effective, minimize the consumption of energy, and complete the jobs within their deadline respectively.

Boyang et al. [21] proposed a resource-aware task scheduling mechanism known as R-Storm which considers both the soft and hard constraints such as number of processing units, bandwidth, and memory respectively. Consequently, the task scheduling is designed as a Quadratic Multiple 3-Dimensional Knapsack problem to balance these three constraints. Their proposed task scheduling algorithm considers inter-rack, inter-node, inter-process, and intra-process communication. Lena Mashayekhy et al. [22] proposed a framework for improving the energy efficiency of MapReduce based big data applications by modeling the energy-aware scheduling as an Integer Programming model and satisfies the Service Level Agreement (SLA). They proposed two heuristic energy-aware MapReduce algorithms namely, EMRSA-I and EMRSA-II. The first one computes the energy consumption rate which is based on the minimum ratio of energy consumption and processing time of tasks when executing on a particular slot and the latter one computes which is based on the average ratio of energy consumption and processing time of tasks when executing on a particular slot respectively. Their proposed algorithms consider the energy efficiency differences of cluster resources and deadline parameter to determine the placement of tasks into the cluster resources.

Peter Bodik et al. [23] proposed a novel deadline-aware scheduling algorithm for the processing of Big data jobs. The main objective of their scheduling is to provide support for both hard and soft deadlines. Their proposed algorithm constructs a Directed Acyclic Graph (DAG) for each job submitted to the system which consists of multiple stages linked by precedence constraints and allocate the resources to the tasks based on the offline allocation model. It schedules the jobs on to C cluster resources within the time slot of 1 to T. Yi Yao et al. [24] proposed a pluggable scheduler known as HaSTE for Apache Yarn [25] which mainly considers the task dependency and resource demand for scheduling of tasks. The main objective of their proposed task scheduling minimizes the makespan of the submitted jobs and increase the utilization of resources. Their proposed scheduling algorithm dynamically schedule the tasks for execution based on the fitness and urgency value of tasks. Here, fitness refers the gap between the resource requirement in the task requests and available resource capacity whereas urgency refers to the property of importance of tasks. The developed aggregate function combines the property of both fitness and urgency value.

IV. TASK SCHEDULING CONSIDERATIONS FOR BATCH AND STREAMING PROCESSING

In Big data, batch processing [26] is an efficient way of processing a large volume of data collected and stored over a period of time whereas streaming refers to the processing of real-time data in an interactive manner. It is important to decide the processing system based on the job requirements, input data (source), and processing time. The Big data stream should continue to process the data streams of online data. The Big data batch processing requires high performance computing cycles whereas the Big data stream processing requires low latency for efficient processing.

A. Task Scheduling Considerations for Batch Processing

Hadoop ecosystem describes that all data should be loaded into Hadoop Distributed File System (HDFS) for processing of batch jobs [27] before starting the execution of a job if there is any change in the data the job has to be executed again. In general, the allocation of resources for batch processing could be done before the execution of tasks which is based on the input data and the task information. Florin Pop and

ValentinCristea [28] explained about the processing of big data as a big batch process by splitting a job into multiple tasks and running on a High Performance Computing (HPC) by distributing the work to the cluster nodes. With batch processing, a single CPU can work on the entire dataset meaning each task will be running on each CPU one after another.

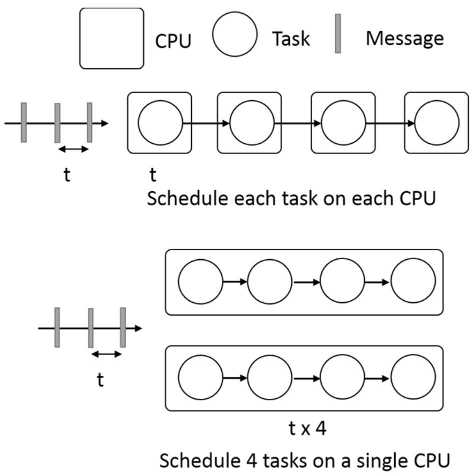

Fig. 1. Top: Stream task scheduled in 4 CPU in a chain. Bottom: All streaming tasks scheduled in a single CPU.

B. Task Scheduling Considerations for streaming processing

In general, the big data platform receives a large amount of streaming data from input data streams such as data sensors, social networking, IoT devices and others. It is difficult to store such large amounts of streaming data hence, it should be processed immediately which requires a lot of computationcycles and memory resources. The scheduling for streaming mainly focus on minimizing latency.Fig. 1 shows a hypothetical example for illustrating the needs of task scheduling for streaming. Here the stream consists of messages coming at 1 msg per t time. We have 4 sequential tasks each requiring t time to process a single message. Lets assume we have 4 CPUs available for processing the messages. As shown in the bottom diagram of fig. 1, if we schedule these 4 task on each of the CPUs, we cannot process the messages with the given rate as it requires $4t$ time. If we schedule the tasks as shown in the top of fig. 1 with each task scheduled to a single CPU, we can achieve the required rate.

Chen Men-meng et al. [29] states that the existing task scheduling algorithms for streaming processing fail to handle the links between the streaming tasks and the dynamic nature of the streaming process and the resources. It is essential to consider the status of the available resource and the demand for the resources while scheduling the streaming tasks. In addition to that, it is important to provide priority to the essential network parameters such as bandwidth and latency. BoyangPeng [30] discussed that the streaming tasks should be scheduled close to the network proximity to avoid the network delay which is communicating with each other. Moreover, task scheduling for streaming tasks is extremely difficult than batch jobs because of the nature of continuous and dynamic nature of input data that requires unlimited processing time. Dawei Sun et al. [31] designed a fault tolerant system which is mainly responsible for guaranteeing the deadline in a big data streaming computing. They stated that fault tolerance is an important metric for achieving the quality of service. Their proposed mechanism allocates the tasks based on the fault tolerance and critical path scheduling technique. Their proposed mechanism solves the trade-off between high fault tolerance and low response time for big data streaming jobs.

V. TASK SCHEDULING IN BIG DATA PLATFORMS OR TOOLS

Apache Hadoop [32] is one of the most popular big data processing frameworks. It has been implemented with the default scheduling policy of FIFO which schedules the jobs coming first and gets higher priority than the later one that leads to starvation of jobs. However, the Fair Scheduling in Hadoop makes an equal share of computing resources among the users or jobs. Delay Scheduling in Hadoop is designed to accommodate changes in the existing MapReduce application and consider the data locality feature to reduce the total execution time of MapReduce application. Capacity Aware Scheduling was introduced by Yahoo with the objective of maximizing the utilization of resources and throughput in a cluster. Apache Spark [33] is an in-memory big data computation framework. It mainly works on the principle of Resilient Distributed Datasets (RDDs) which has been implemented both the static and dynamic task scheduling algorithms for those RDDs. The fair scheduling policy in Spark group the jobs into pools and assign weights into each pool. The dynamic resource allocation policy allocates the resources to the jobs based on the workload of the cluster resources in a dynamic manner.

Apache Mesos [34] is a container based cluster resource management framework which is implemented with a finegrained Dominant Resource Fairness (DRF) algorithm that allocates the sharing of resources across the applications running on the platform. It decides a number of resources to be allocated to each framework and provides the resource offers to the schedulers. It is able to achieve near-optimal data locality and better scalablility based on their fine-grained resource allocation mechanism. However, it may fail to consider the resource requirements of applications running on Mesos. Apache Flink [6] is implemented with an immediate scheduling and queued scheduling algorithm which returns the slot immediately and queues the request and returns the slot whenever it is available respectively. Apache Storm [35] is implemented with a default round-robin scheduling for the placement of streaming tasks on the execution nodes. It does not consider the availability of the resource or the applications resource requirement while scheduling the tasks, this may lead to under-utilization or over-utilization of the resources. Apache Heron [7] is implemented with Round Robin and First Fit Bin Decreasing packing plan algorithms for scheduling the streaming applications in the big data processing. Similar to Apache Storm task scheduling algorithms, these algorithms dont consider the resource requirement and the resource availability which leads to under-utilization or over-utilization of resources.

Derek G. Murray et al. [36] designed a timely dataflow system known as Naiad for executing data parallel and cyclic dataflow applications. It provides the high throughput for batch processing and low latency for stream processing. Also, it supports both iterative and incremental based computing approaches. It is embedded with a timely dataflow computation model which enhances dataflow computation with time-stamps and provides the base for an efficient, lightweight coordination mechanism. It provides the support for constructing various high-level programming models on top of Naiads lowlevel primitives which enable streaming data analysis, iterative machine learning and interactive graph mining. Leonardo Neumeyer et al. [37] designed an S4 (Simple Scalable Streaming System) model known as Yahoo S4 which is based on the MapReduce model for solving the real-world problems. It is implemented with data mining and machine learning algorithms. MicrosoftsTimeStream [38] is a distributed system which is specifically designed for continuous processing of low latency streams. It is designed based on the MapReduce-style batch processing model. Also, it is embedded in an abstraction unit known as resilient substitution to handle the recovery of failure and dynamic reconfiguration according to the load.

Dryad [39] is a distributed execution engine for achieving high performance and running coarse grain data parallel applications. Quincy scheduler [40] is integrated with Dryad that aims to target the task level scheduling in computing clusters. It converts the scheduling problem into a graph-based structure and handles the conflict between data locality and fairness. It encodes both the network structure and the tasks which are in waiting state and solved the scheduling problem using mincost flow solver. In addition to that, it provides the support for more sophisticated scheduling policies but, sometimes it is not suitable for shorter workload type of jobs. Kay Ousterhout et al. [41] designed a stateless distributed scheduler named Sparrow which adapts the power of two balancing techniques [42] for parallel task scheduling. It supports both per job and per task-level constraints. It is implemented with two allocation policies such as strict priorities and weighted fair sharing. It supports various applications which can run on Hadoop and Spark platforms. The main challenge in Sparrow is balancing the load between the distributed schedulers and reducing the response time. Sparrow allows to distribute the workload of the resource but, it doesn’t consider the availability of the resource while making scheduling decisions which may overload the resources.

Aurora [43] is a Mesos framework or a service scheduler running on top of Apache Mesos. It facilitates to run longrun jobs, cron jobs, and adhoc jobs. In general, Mesos is concerned about individual tasks whereas a job consists of multiple task instances. However, an Aurora job consists of a task template and instructions for creating task instances. In summary, Aurora is responsible for handling jobs consists of multiple tasks whereas Mesos is responsible for handling tasks comprises of multiple processes. It creates a sandbox for each task when it starts that would be garbage collected when the task finishes its execution. Marathon [44] is a container orchestration platform for Mesos and DataCenter Operating System (DC/OS). It is the first framework which is capable to run directly on top of Mesos. It’s scheduler processes can be directly initiated on Mesos framework. It is also a coercive tool to run other frameworks like Chronos (A distributed and faulttolerant scheduler) [45] and it has the ability of dynamically placing the containers.

REEF (Retainable Evaluator Execution Framework)[46] provides a control plane to schedule and coordinate data plane on cluster resources for data processing applications. It is independent from the specific programming which provides an application framework that helps to develop and execute the analytic tools in a cluster. It provides the ability for key abstractions such as Driver, Task, Evaluator, and Context. The Driver is responsible for the implementation of allocating resources and scheduling the tasks. The Task is the smallest unit of code to be considered for execution in an Evaluator. The Evaluator is a runtime environment which retains the containers state to avoid resource allocation and scheduling costs and the Context is a state management environment. It hides the underlying details of the resource manager into an Environment adapter layer that translates the requests into underlying resource manager actions. It is responsible for simplying the process of communicating the Driver and Task components in a large-scale data processing application.

VI. OVERVIEW OF TWISTER2 AND PROPOSED TASK SCHEDULING MODEL

A. Overview of Twister2

SupunKamburugamuwa Geoffrey Fox proposed a Big data programming toolkit named Twister2 [8] empowered with the dataflow programming model. It hides the underlying details of communication, synchronization, and Input and Output operations. It is purely designed based on an event-driven model for data processing which has been designed with clear functional layers of communication, resource scheduling, task execution, data abstractions and fault-tolerance mechanism. It is designed to handle different kind of applications including batch, stream, and Microservices.

B. Task Scheduling Model

We have extended the scheduling model of [47] for task scheduling in Big data which comprises of Job Model, Resource Model, Performance Metrics, Scheduling Policy, and Programming Model as explained below.

- Job model - It provides the abstraction of jobs (consists of multiple tasks) and its requirements. The requirements are classified into hard and soft constraints. The job model handles different type of jobs namely batch, streaming, MPI, and microservices.
- Resource model - It describes the characteristics and performances of data centers, hosts, rack, and network links. The resource model maintains the metadata that contains resource characteristics in terms of properties and value.

- Scheduling policy/algorithm - The scheduling policy or algorithm implemented in Task Scheduler which is based on specific goals such as optimization of total computational time or utilization of cluster resources or both. It operates based on the characteristics of job attributes, resource constraints, resource workload, and input data. The scheduling algorithm is mainly categorized into static and dynamic task scheduling algorithms.

- Performance metrics - It is used to evaluate the performance improvements gained by the proposed task scheduling model. In our proposed work, we have considered and evaluated the various performance metrics as listed in table II.

- Programming model - The programming model is helpful for providing the interface to the scheduler. In this proposed approach, we have made use of dataflow programming model for interfacing the task scheduler with other components.

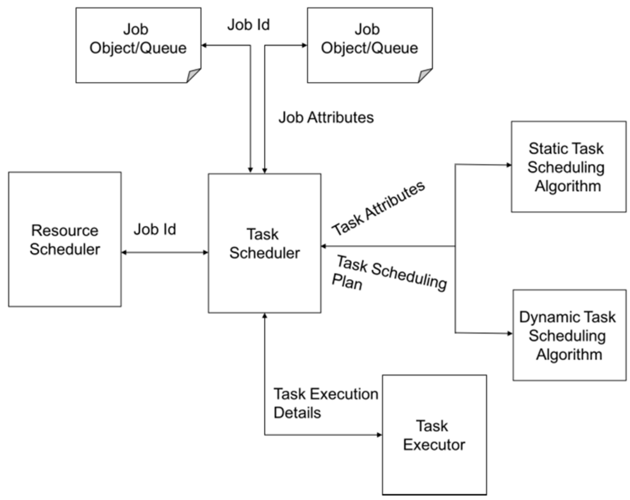

Fig. 2. Task Scheduling in Twister2

C. Task Scheduling System in Twister2

The task scheduling system is designed such that it is able to handle various type of jobs discussed above. Also, it facilitates to schedule both static and dynamic task graphs. The workflow functionality of proposed task scheduler is shown in Fig. 2. It is integrated with static as well as dynamic task scheduling algorithms which invokes an appropriate algorithm based on the application, input data, and the input data source. The Resource Scheduler sends the job id to the Task Scheduler for generating the task schedule plan. The Task Scheduler receives the job id and fetches the corresponding job attributes from the job object and the task scheduling policy from the scheduling configuration file. First, it computes the number of task instances to be created for the execution of a job which is based on the parallelism of the number of tasks. Subsequently, it generates the task schedule plan which consists of the number of containers to be created and the task instances to be hosted in the containers. Finally, it sends the task execution details to the Task Executor for the execution of tasks on the worker nodes.

VII. Summary of Findings and Research Directions

In this research paper, we studied the state-of-art of various task scheduling algorithms proposed by the researchers, task scheduling systems in popular big data platforms, and task scheduling scenarios to be considered for big data batch processing and streaming processing. Based on the literature review, the future research directions are classified into three types which could be addressed by our proposed task scheduling system.

A. Future Research Directions for Supporting both Static and Dynamic Task Scheduling

The static and dynamic task scheduling depend on the nature of the job and the input data. However, both static and dynamic task scheduling is suitable for batch processing and stream processing of big data jobs. Hence, the proposed task scheduling would be accommodated with both the scheduling mechanism which is shown in Table I.

B. Future Research Directions for Handling Different Type of Jobs

The big data platform should be able to handle different type of jobs for processing big data. Hence, the task scheduling in the big data platform should facilitate the scheduling mechanism for effectively schedule those jobs. The table I represents the various task scheduling algorithms and supported job types in the various big data platforms. It infers that the proposed task scheduling system is accommodated with both static and dynamic task scheduling algorithms and it is able to schedule or manage four kinds of jobs namely MapReduce, streaming, MPI and Micro Services.

C. Future Research Directions for Considering Different Type of Performance Factors

In order to consider the various scenarios discussed above, the proposed task scheduling system has been designed to accomplish the various objectives that reduce the total computation time and increase the resource utilization in a near optimal manner. It considers the various essential factors namely data locality, resource workload, energy consumption, job attributes, deadline of the job, etc. From this study, it is identified that the existing task scheduling techniques are either focused on user-centric or resource-centric, and they failed to address both the factors at once. However, the proposed task scheduling system first classifies the performance factors into user-centric and resource-centric, and considers both the factors as shown in table II along with their definition. It also infers that the proposed task scheduling system is able to resolve the problems that persist in existing task scheduling techniques by considering both the essential user-centric and resource-centric parameters.

VIII. Conclusion

Task scheduling in Big data is one of the active research areas which plays a major role in the completion of Big data processing and effectively utilize the cluster resources. From the literature review, it has been identified that there is no common task scheduling model to accommodate both the static and dynamic task graphs and handle different type of jobs as discussed above. Hence, the proposed task scheduling model is able to handle both static and dynamic task scheduling. Additionally, it has the ability to schedule different types of jobs namely batch, streaming, MPI, and micro-services which are missing in other existing works. Furthermore, the proposed task scheduling model considers both the user-centric and resource-centric performance factors. The future work will explore (i) to introduce fault-tolerance in task scheduling and (ii) performs various performance testing through popular benchmarks and publish the results.

TABLE I. TASK SCHEDULING ALGORITHMS AND SCHEDULING JOB TYPES

Big Data Platforms	Scheduling Types		Scheduling Job Types				Dataflow Programming Model
	Static	Dynamic	Batch	Streaming	MPI	FaaS / Microservices	
Spark	No	Yes	Yes	Yes	No	No	Yes
Flink	Yes	No	Yes	Yes	No	No	Yes
Heron	Yes	No	No	Yes	No	No	Yes
Storm	Yes	No	No	Yes	No	No	Yes
Hadoop	Yes	No	Yes	No	No	No	No
Twister2	Yes	Yes	Yes	Yes	Yes	Yes	Yes

TABLE II. TASK SCHEDULING PERFORMANCE FACTORS

Performance Factors	Description	User / Resource Centric
Deadline	Reduce the maximum time to be taken to complete the user application/job.	User-Centric
Execution Time	Minimize the time taken to complete the actual execution of a task or a job.	User-Centric
Completion Time	Minimize the time taken to complete the execution of job which consists of both execution and communication time.	User-Centric
Makespan	Minimize the total time taken to complete a particular job which consists of multiple map and reduce tasks.	User-Centric
Data Locality	Minimize the distance between the input data node and the actual execution node.	User-Centric
Resource utilization	Minimize the utilization of cluster resources.	Resource-Centric
Energy Consumption	Minimize the consumption of energy in the cluster resources.	Resource-Centric
Fault Tolerance	Minimize the failure of jobs and job executors/job managers	Resource-Centric

ACKNOWLEDGMENT

This work was partially supported by the Indiana University Precision Health Initiative and by NSF CIF21 DIBBS1443054. We thank Intel for their support of the Juliet system, and extend our gratitude to the Future Systems team for their support with the infrastructure.

REFERENCES

[1] M. D. Assuno, R. N. Calheiros, S. Bianchi, M. A. Netto, and R. Buyya, “Big data computing and clouds: Trends and future directions,” *Journal of Parallel and Distributed Computing*, vol. 79-80, no. Supplement C, pp. 3 – 15, 2015, special Issue on Scalable Systems for Big Data Management and Analytics. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514001452

[2] R. Magoulas and L. Ben, “Big data processing: Big challenges and opportunities,” 2011.

[3] N. Samal and N. Mishra, “Big data processing: Big challenges and opportunities,” Journal of Computer Sciences and Applications 3(6):177180, 2015.

[4] C. Lam, *Hadoop in action*. Manning Publications Co., 2010.

[5] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud’10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863103.1863113

[6] 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.

[7] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, “Twitter heron: Stream processing at scale,” in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’15. New York, NY, USA: ACM, 2015, pp. 239–250. [Online]. Available: http://doi.acm.org/10.1145/2723372.2742788

[8] S. Kamburugamuve and G. Fox, “Designing twister2: Efficient programming environment toolkit for big data,” http://dsc.soic.indiana.edu/publications/Twister2.pdf, 2017, Technical Report.

[9] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernandez-´ 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.

[10] J. Conejero, S. Corella, R. M. Badia, and J. Labarta, “task-based programming in compss to converge from hpc to big data.”

[11] F. Juarez, J. Ejarque, and R. M. Badia, “Dynamic energyaware scheduling for parallel task-based application in cloud computing,”

Future Generation Computer Systems, vol. 78, no. Part 1, pp. 257 – 271, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X1630214X

[12] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken, "Legion: Expressing locality and independence with logical regions," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 66:1–66:11. http://dl.acm.org/citation.cfm?id=2388996.2389086

[13] S. Treichler, M. Bauer, and A. Aiken, "Realm: An eventbased low-level runtime for distributed memory architectures," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, ser. PACT '14. New York, NY, USA: ACM, 2014, pp. 263–276. http://doi.acm.org/10.1145/2628071.2628084

[14] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 323–336. http://dl.acm.org/citation.cfm?id=1972457.1972490

[15] "Capacity Scheduler," https://hadoop.apache.org/docs/r2.8.0/hadoopyarn/hadoop-yarn-site/CapacityScheduler.html, accessed: 2017-Sep-29.

[16] Y. Wang, R. Rao, and Y. Wang, "A round robin with multiple feedback job scheduler in hadoop," in *2014 IEEE International Conference on Progress in Informatics and Computing*, May 2014, pp. 471–475.

[17] B. Jiang, J. Wu, X. Shi, and R. Huang, "Hadoop scheduling base on data locality," *CoRR*, vol. abs/1506.00425, 2015. [Online]. Available: http://arxiv.org/abs/1506.00425

[18] Y.-C. Kao and Y.-S. Chen, "Data-locality-aware mapreducerealtime scheduling framework," *Journal of Systems and Software*, vol. 112, no. Supplement C, pp. 65 – 77, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121215002344

[19] "First Fit DecreasingPacking Algorithm," https://twitter.github.io/heron/docs/developers/packing/ffdpacking/.

[20] C. He, Y. Lu, and D. Swanson, "Matchmaking: A new mapreduce scheduling technique," in *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 40–47. [Online]. Available: https://doi.org/10.1109/CloudCom.2011.16

[21] P. Boyang, H. Mohammad, and H. Zhihao, "R-storm: Resource-aware scheduling in storm," ser. Annual Middleware Conference. ACM, 2015. [Online]. Available: http://dx.doi.org/10.1145/2814576.2814808

[22] L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang, and W. Shi,"Energy-aware scheduling of mapreduce jobs for big data applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2720–2733, Oct 2015.

[23] P. Bodik, I. Menache, J. S. Naor, and J. Yaniv, "Brief announcement: Deadline-aware scheduling of big-data processing jobs," in *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '14. New York, NY, USA: ACM, 2014, pp. 211–213. [Online]. Available: http://doi.acm.org/10.1145/2612669.2612702

[24] Y. Yao, J. Wang, B. Sheng, J. Lin, and N. Mi, "Haste: Hadoop yarn scheduling based on taskdependency and resource-demand." in *IEEE CLOUD*. IEEE, 2014, pp. 184–191. [Online]. Available: http://dblp.unitrier.de/db/conf/IEEEcloud/IEEEcloud2014.htmlYaoWSLM14

[25] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar,R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in Proceedings of the 4th Annual Symposium on Cloud Computing, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523633.

[26] "Batch Processing vs Stream Processing," https://www.thomashenson.com/ultimate-big-data-battle-batchprocessing-vs-streaming-processing, accessed: 2017-Oct-22.

[27] "Batch Processing," https://www.datasciencecentral.com/profiles/blogs/batchvs-real-time-data-processing, accessed: 2017-Oct-22.

[28] C. V. Pop F, The art of scheduling for big data science. Chapman Hall/CRC Big Data Series, 2015.

[29] M. Chen, C. Zhang, Z. Li, and K. Xu, "A task scheduling approach for real-time stream processing," in International Conference on Cloud Computing and Big Data, CCBD 2014, Wuhan, China, November 12-14, 2014, 2014, pp. 160–167. [Online]. Available: https://doi.org/10.1109/CCBD.2014.22

[30] "Elasticity and resource aware scheduling in distributed data stream processing systems," http://hdl.handle.net/2142/78453, accessed: 2017Oct-24.

[31] D. Sun, G. Zhang, C. Wu, K. Li, and W. Zheng, "Building a fault tolerant framework with deadline guarantee in big data stream computing environments," Journal of Computer and System Sciences, vol. 89, no. Supplement C, pp. 4 – 23, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000016301143

[32] T. White, Hadoop: The Definitive Guide, 1st ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2009.

[33] M. Zaharia, D. Borthakur, J. SenSarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in Proceedings of the 5th European Conference on Computer Systems, ser. EuroSys '10.New York, NY, USA: ACM, 2010, pp. 265–278. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755940

[34] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308. http://dl.acm.org/citation.cfm?id=1972457.1972488

[35] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donhamet al., "Storm@ twitter," in Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014, pp. 147–156.

[36] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, "naiad: A timely dataflow system."

[37] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in 2010 IEEE International Conference on Data Mining Workshops, Dec 2010, pp. 170–177.

[38] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, and Z. Zhang, "Timestream: Reliable stream computation in the cloud," in Proceedings of the 8th ACM European Conference on Computer Systems, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp.1–14 http://doi.acm.org/10.1145/2465351.2465353

[39] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, ser. EuroSys '07. New York, NY, USA: ACM, 2007, pp. 59–72. [Online]. Available: http://doi.acm.org/10.1145/1272996.1273005

[40] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 261–276. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629601

[41] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in Proceedings of the TwentyFourth ACM Symposium on Operating Systems Principles, ser. SOSP '13. New York, NY, USA: ACM, 2013, pp. 69–84. http://doi.acm.org/10.1145/2517349.2522716

[42] M. Mitzenmacher, "The power of two choices in randomized load balancing," IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 10, pp. 1094–1104, Oct 2001.

[43] "Aurora Scheduler," http://aurora.apache.org/documentation/latest/, accessed: 2017-Oct-30.

[44] "Marathon Container Orchestration Platform," https://mesosphere.github.io/marathon/, accessed: 2017-Oct-30.

[45] "Chronus Fault Tolerant Scheduler," https://github.com/mesos/chronos/, accessed: 2017-Oct-30.

[46] B.-G. Chun, T. Condie, Y. Chen, B. Cho, A. Chung, C. Curino,

[47] C. Douglas, M. Interlandi, B. Jeon, J. S. Jeong, G. Lee, Y. Lee, T. Majestro, D. Malkhi, S. Matusevych, B. Myers, M. Mykhailova, S. Narayanamurthy, J. Noor, R. Ramakrishnan, S. Rao, R. Sears, B. Sezgin, T. Um, J. Wang, M. Weimer, and Y. Yang, "Apache reef: Retainable evaluator execution framework," ACM Trans. Comput. Syst., vol. 35, no. 2, pp. 5:1–5:31, Oct. 2017. [Online]. Available: http://doi.acm.org/10.1145/3132037

[48] T. S. Somasundaram and K. Govindarajan, "Cloudrb: A framework for scheduling and managing high-performance computing (hpc) applications in science cloud," Future Generation ComputerSystems, vol. 34, no. Supplement C, pp. 47 – 65, 2014, special Section: Distributed Solutions for Ubiquitous Computing and Ambient Intelligence. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13002884