**International Journal of Scientific Research in Computer Science and Engineering** **Research Paper**
Vol.5, Issue.5, pp.10-15, October (2017) **E-ISSN:** 2320-7639

# Performance and Analysis of Flow between Annular Space Surrounded by a Rotating Coaxial cylinder with Co-axial Cylindrical Porous Medium

**Santosh Patil[1*], Ediga Lingappa[2], Mothe Rakesh[3]**

[1*]Dept. of Computer Science and Engineering, Institute of Aeronautical Engineering, Hyderabad, India
[2]Dept. of Computer Science and Engineering, Institute of Aeronautical Engineering, Hyderabad, India
[3]Dept. of Computer Science and Engineering, Institute of Aeronautical Engineering, Hyderabad, India

**Corresponding Author: santosh.p.shr@gmail.com*

**Available online at: www.isroset.org**

Received 11th Sep 2017, Revised 14th Sep 2017Accepted 19th Oct 2017, Online 30th Oct 2017

***Abstract***— Due to latencies, the hadoop map reduce are complicated to scale to multiple clouds. Because of latencies, the hadoop map reduce are difficult to scale to multiple clouds. Because of this problem, to improve the performance at variable load, it provides over-provisioning in internal cloud. Here we propose a Bstream - cloud bursting framework. It consists of two major features. They are Stream-processing in the external cloud. Hadoop in the internal cloud. These two features are used to realize inter-cloud map reduce. Stream processing in external cloud enables parallel uploading; processing and also parallel downloading of data can minimize network latencies. It guarantees service-level objective (SLO) of meeting job deadlines.

***Keywords***— Bstream, Map Reduce, stream processing.

## I. INTRODUCTION

In recent years, Hadoop Map Reduce is used extensively in near real-time “Big Data” analytics like advertising, traffic log analysis, sentiment analysis, and many. These applications include severe service-level objectives such as deadline and experience highly changeable input load. Mislaid SLOs owing to high load can result in important penalties which include; updating customer facing content will be delayed, associated loss in revenue, and so forth. Such situation will be handled by a private data center is which is typically more-furnished; it may causes wastage of resources.

Cloud bursting is an option which is alternative to over provisioning by off-load the surplus load from the internal cloud (IC) to an external cloud (EC) (e.g., Amazon EC2, Rack space). However, it includes two main difficulties in scaling Hadoop Map-Reduce with cloud bursting. First, Hadoop being a batch-processing system, require the intact input data for the job must be materialized before the start of computation. As in inter-cloud data transfer latencies are at slightest an order of magnitude greater than that within a data center, batch processing using Hadoop in EC incurs enormous startup latencies—stand by for the complete input data to be uploaded, and later processing it. The second complexity arises because of condensed coupling of shuffle phase in Map-Reduce with reducers. Shuffle phase only starts after reducers have started (Occupied slots). As shuffle requires all-all node communication, data will be shuffled from EC to reducers in IC, which takes prolonged due to difference between intercloud and intra-cloud bandwidth. This extended shuffle

Phase delays job completion time and also causes idle cpu cycles within reducers. Mappers in EC to perform streaming read from remote HDFS in IC which minimizes startup latencies. They also propose techniques to reduce the effect of elongated shuffle phase. But these jobs uses on previous version of Hadoop(v1.0) that inherently causes idle cpu cycles due to fixed partitioning of map and reduce slots on a node. Thus, even if reduce slots are idle, a job cannot use them for map tasks.

YARN, 2 is newer version of hadoop, in where fixed partitioning of slots is no longer used. Now these slots are free we can use these slots for performing map tasks, by delaying the start of reducers. So slots are efficiently used, thereby job completion time will be reduced. Start of reducers cannot be delayed, until the shuffle is decoupled from reducers in IC, because inter-cloud setting with elongated shuffle. The shuffle from EC can be further enhanced by performing reduce operation in EC that reduces data download size.

Bstream is a new cloud bursting framework is proposed in this project, that to address the above difficulties. Bstream uses stream processing in EC and YARN is used in IC. Using Storm, on the incoming stream of data both map and reduce operations execute as and when it arrives in EC. This minimizes startup latencies in EC by overlapping processing with input data transfer. By executing reduce action in EC, it minimizes the download size. Using check pointing strategies shuffle phase is, that download intermittent reduces output from EC to reducers in IC. Bstream allow parallel uploading, processing and downloading of data by using stream processing and check pointing strategies. We currently consider meeting deadlines for individual jobs whose input data is initially present only in IC. Such a scenario is encountered by enterprises that use their private data center (IC) for normal operation and employ cloud bursting only when there are load surges. Extending the framework to multiple jobs is part of future work.

In this paper we addressed the implementation of an Efficient method for word count across external cloud and internal cloud. Here we consider data file as job and make use of stream processing and check pointing strategies. Bstream make use of a stream processing in External cloud (EC) and YARN in will be used Internal cloud (IC)**.** On incoming data storm map and reduce operations over are executed; Check pointing strategies are used to enable pipelined uploading, processing and downloading of data.

## II. REVIEW OF LITERATURE

Herodotos Herodotou, Harold L, *et al.*[1], they propose a novel Starfish approach, which builds on Hadoop whereas adapting to user requirements and system workloads to provide outstanding performance, with no need for users to understand and handle the many tuning knobs in Hadoop. Whereas Starfish's system architecture is guide the work on self-tuning database systems. Starfish's tuning goals and solutions are related to projects like Hive, MRShare, Pig, Quincy, and Scope. The novelty in Starfish's approach describes about how it focuses concurrently on different workload granularities.

Christian Vecchiola, Rodrigo N. *et al.*, [2] , proposed a technique, Aneka's deadline-driven provisioning mechanism responsible for supporting QoS-aware implementation of scientific applications in hybrid clouds. Aneka is a software platform and which is development of distributed applications in the cloud. Which uses the enumerate resources of a various network of workstations, clusters, and data centers, on demand. Platform as a Service model is implemented in Aneka. System administrators influence a collection of tools to supervise and control the cloud. Aneka assign resources from different sources in order to decrease application execution time. Some improvements are required in Aneka's dynamic resource provisioning, are underneath development, and one time these improvements are accessible, we be expecting that applications will run much efficiently in hybrid resources.

Tekin Bicer, David C, *et al.*, [3], In this paper, they illustrate a software framework which enables data rigorous computing by cloud bursting, i.e., through a amalgamation of work out resources from a local group and a cloud to carry out Map-Reduce type processing. Here author chronicle a middleware which backing Map-Reduce style API, where the data will be circulated across a local cluster and a cloud. Data at one end is handled using computing resources at another end, by doing this we can achieve faster execution i.e., work stealing.

Sriram Kailasam, Nathan Gnanasambandam, *et al.*,[4], In this paper, They study the viability of cloud bursting for data-intensive workloads. Here they assume that the data necessary for computation is present in a single cloud. Based on workload characteristics and the operating environment (node and network bandwidth), also characterize the operational regimes of cloud bursting into stabilization mode and acceleration mode. As the workload characteristics change from data-intensive to compute-intensive, the operating mode shifts from stabilization mode to acceleration mode.

Kamal Kc, Kemafor Anyanwu *et al.*, [5], In this paper, they expand real time cluster scheduling method to account for the two-phase computing way of Map Reduce. In existing cloud-based data processing environments User constraints such, which are very important requirements, are not considered. Here author describes, benchmark for scheduling jobs based on user specified constraints and preliminary valuation of a Deadline pressure Scheduler, that assures, scheduled of execution of those jobs whose deadline can met. The main drawback is that Author left out some aspects of Constraint Scheduler such as map/reduce task runtime estimation, filter ratio estimation, data distribution and multiple Map Reduce cycle support.

Michael Mattess, Rodrigo N. *et al.*, [6], propose a novel policy that accelerate execution of deadline-constrained MapReduce applications, by allowing parallel execution of tasks, in sequence to meet up deadline for fulfillment of the Map phase of the application. They offered a dynamic provisioning policy for MapReduce applications and a prototype implementation of the correspondent system in the Aneka Cloud Platform. Proposed policy was capable to convene deadlines of applications. But it's difficult to scale the policy to optimize the provisioning for more complex scenarios, such as multiple independent applications and composite MapReduce applications, where one application consumes the output of a previous application.
Yuan Luo, Zhenhua Guo, *et al.*, Judy Qiu1, Wilfred Li [7], In this paper, they present a hierarchical MapReduce framework which collect computation resources from

various clusters and run MapReduce jobs on them. The global controller in their framework divide the data set and distribute them to multiple "local" MapReduce clusters, and based on capabilities of each cluster, workload will be assigned. they propose a hierarchical MapReduce framework which collect confined cluster resources into a further proficient one for running MapReduce jobs. MapReduce jobs grouped into four groups. In our framework map-intensive computation achieved by distributing map reduce jobs to, map-only and map-mostly categories and where global node is responsible for collection and combing of outputs. Here author fails to address data locality issue of input dataset.

## III. METHODOLOGY

Figure 1 gives the System Architecture of extending map reduce across clouds with Bstream. The architecture is divided into three parts; they are data input phase, processing phase and result phase. The user submits the MapReduce job to the controller along with its deadline. The controller uses the estimator to determine the resource allocation. The estimator refers the job profile database and uses the analytical model to estimate the resource allocation in IC. If the estimated resource allocation returned to the controller by the estimator exceeds the available resources in IC, then the controller initializes the burst coordinator with number of maps to burst and the time to start bursting. The controller submits the MapReduce job to the Hadoop framework. Then coordinator splits the jobs, these jobs are evenly distributed to both internal cloud and external cloud, in external cloud initially in external cloud first it apply then followed by mapping then output is sent to internal cloud, at the same time in internal cloud mapping and reducing operation is applied, then finally output from both external cloud and internal cloud are combined to produce final out as word count.

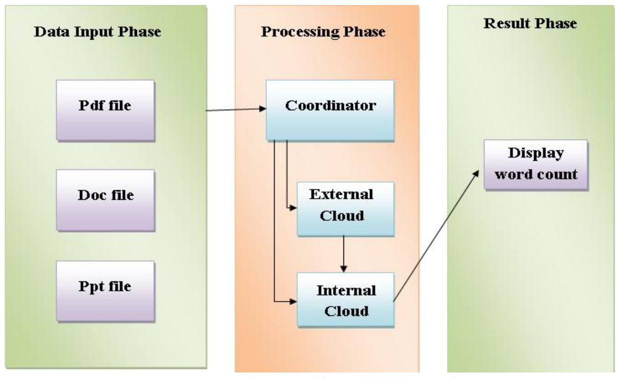


Figure 1: System Architecture for an efficient method for word count across EC and IC.

The time spent in downloading the output from EC can contribute a significant portion to the overall completion time of the burst job if the output to input ratio is high. With stream processing in EC, partial outputs are available at the reducers as soon as some data is processed through the Storm topology. We use check pointing to overlap output transfer with input transfer/processing, thus minimizing the overall completion time of the job.

The flowchart of the proposed system is as shown in Figure 2. The input is provided by user; here we consider the files like pdf, ppt, doc etc as input file. And then the controller uses the estimator to determine the resource allocation. The estimator refers the job profile database and uses the analytical model to estimate the resource allocation in IC. If the estimated resource allocation returned to the controller by the estimator exceeds the available resources in IC, then Burst coordinator divides the job i.e. splits the file, and equally distributes these files to external cloud and internal cloud for processing, then the output from external cloud is sent to internal cloud to produce combined output. If the resources in internal cloud not exceeds, the processing task is only assigned to internal cloud, finally internal cloud returns the word count as output.

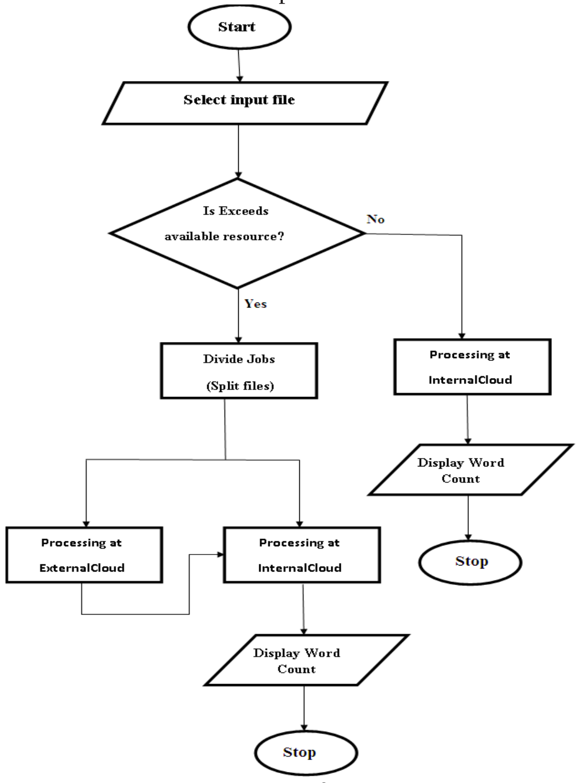

Start
Select input file
Is Exceeds available resource?
No
Yes
Divide Jobs (Split files)
Processing at InternalCloud
Display Word Count
Stop
Processing at ExternalCloud
Processing at InternalCloud
Display Word Count
Stop

Figure 2: Flow chart of Bstream System

In check pointing, we take a snapshot of the current output of a reducer and transfer the snapshot data to IC. Figure. 3 present the check pointing strategy in detail. The partial outputs from reducer are stored in LevelDB. The data output rate from Storm reducers is higher than the rate at which data can be downloaded. Therefore, a key (present in the output snapshot) can get updates from Storm reducer during checkpoint transfer.

Such an updated key will require retransmission in the next checkpoint. To reduce the probability of retransmission, the checkpoint controller employs continuous updating strategy, where it checks for updates before sending a key to reducers in IC. If a key has received updates, then that key is not transmitted as part of the current checkpoint, thus reducing the retransmission overhead.

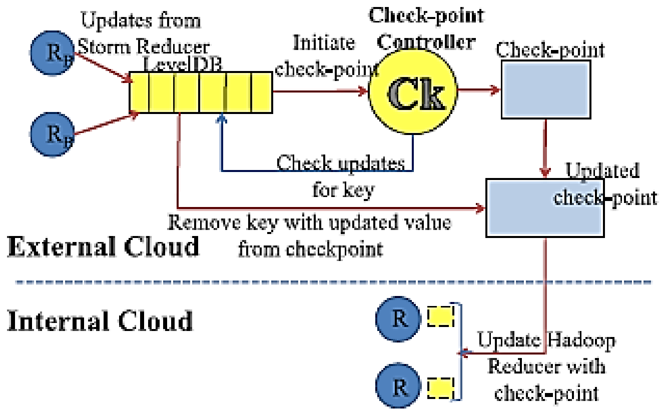


Figure 3: Checkpointing strategy used in External cloud

If the key does not receive any further updates, it will be transmitted as part of the next checkpoint. Since the network is kept busy throughout, there is no loss in terms of bandwidth utilization. If a certain key gets updated after it between its output value and checkpoint value are transmitted as part of the next checkpoint. The next checkpoint also includes keys that newly arrived in the interval between the two checkpoints. Depending on the characteristics of the data set and the size of data processed so far, the probability of updates or addition of new keys keeps varying.

Figure. 4 illustrate different ways to split a MapReduce job across multiple clouds. In Fig. 4a, the map tasks are distributed across IC and EC, while the reduce tasks are executed only in IC. This results in huge data transfer overheads during shuffle operation. This could be overcome

By splitting the MapReduce job into different sub-jobs and executing them separately on IC and EC (refer Figure. 1b).A final MapReduce job (G) is used to merge the results of these jobs. However, this approach introduces the overhead of launching an\additional job for performing global reduction. Figure.4c avoids this by distributing both map and reduces tasks across multiple clouds. Also, the final result from reducers in EC has to be downloaded to IC. In Bstream, we adopt an approach illustrated in Fig. 1d wherein the output of MapReduce job in EC is directly transferred to the reducers in IC. We propose a model to determine the start time of reducers in IC, considering the overheads for inter-cloud and intra-cloud data transfer. This minimizes wastage of compute cycles during shuffle phase without incurring the overhead of global reduction.

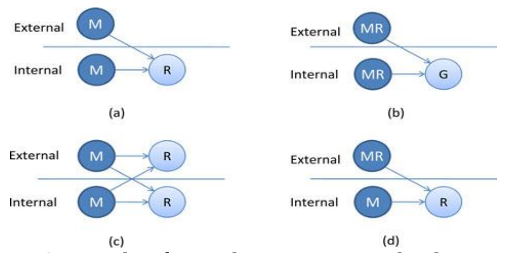


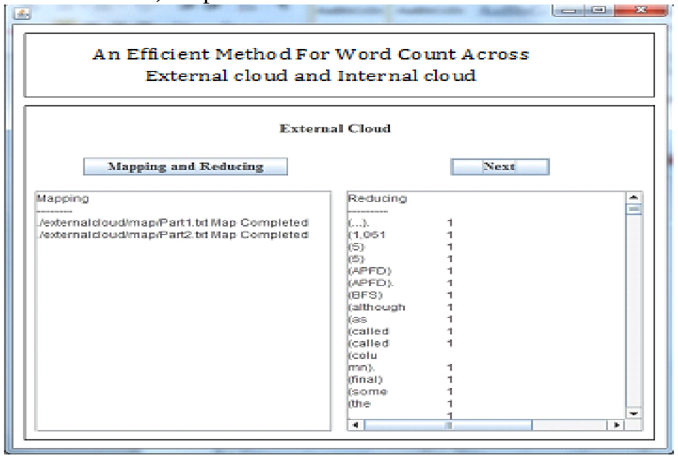
Figure 4: Approaches for implementing inter-cloud MapReduce.

## IV. RESULT

In our proposed Bstream, a new cloud bursting framework to address the above difficulties. BStream uses stream processing engine called Storm3 in EC and YARN in IC. Using Storm, both map and reduce operations execute on the incoming stream of data as and when it arrives in EC. This overlaps processing with input data transfer, thus minimizing startup latencies in EC. By executing reduce operation in EC, it decreases the download size. Shuffle from EC is also decoupled from YARN reducers, allowing the reducers to start much later in the job lifecycle.

Shuffle is further optimized using checkpointing strategies that download intermittent reduce output from EC to reducers in IC. Thus, using stream processing and checkpointing strategies, Bstream enables parallel uploading, processing and downloading of data. BStream uses an analytical model to estimate what portions of MapReduce job to burst, when to burst and when to start the reducers, to meet job deadline. We currently consider meeting deadlines for individual jobs whose input data is initially present only in IC. Such a scenario is encountered by enterprises that use their private data center (IC) for normal operation and employ cloud bursting only when there are load surges.

Figure 5 shows the output of external cloud where initially words sorted in ascending then word count are displayed then output of this is sent to internal cloud., similarly Figure 6 shows the output of internal cloud, words are sorted in initially sorted in ascending then word count is displayed. Figure 7 shows final output which displays the word counts, initially mapping and reducing operations are applied on jobs i.e. file during mapping process the data in file are sorted in ascending order and in reduce phase the word count are displayed. Jobs are distributed across internal and external cloud, output of external cloud.



An Efficient Method For Word Count Across
External cloud and Internal cloud

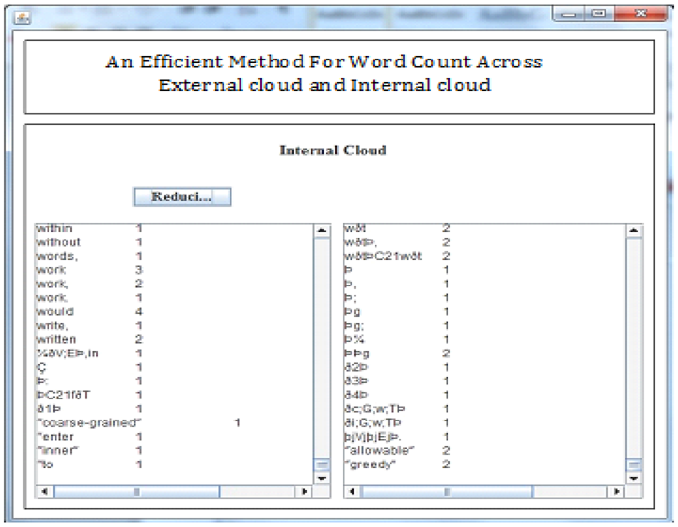
External Cloud

Mapping and Reducing    Next

```
Mapping
--------
./externalcloud/map/Part1.txt Map Completed
./externalcloud/map/Part2.txt Map Completed
```

```
Reducing
--------
(...).          1
(1,061          1
(5)             1
(5)             1
(APFD)          1
(APFD).         1
(BFS)           1
(although       1
(as             1
(called         1
(called         1
(colu
mn).            1
(final)         1
(some           1
(the            1
                1
```

Figure 5: Word count processing in External cloud



An Efficient Method For Word Count Across
External cloud and Internal cloud

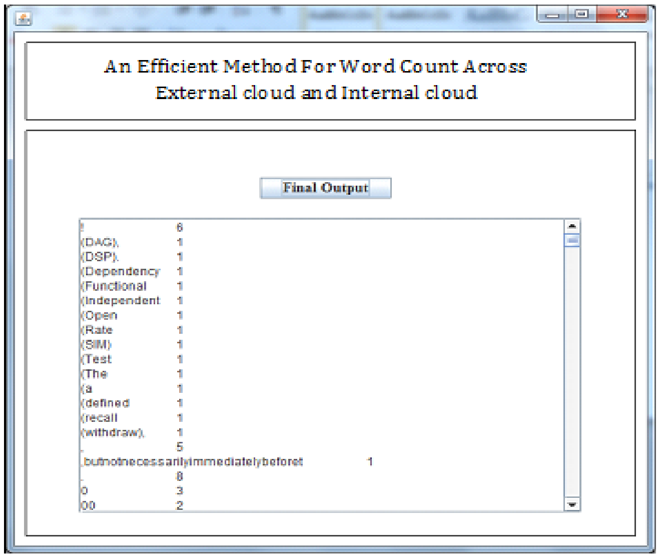
Internal Cloud

Reduci...

```
within          1
without         1
words,          1
work            3
work,           2
work.           1
would           4
write,          1
written         2
¾ðV;Eÿ,in       1
Ç               1
Þ:              1
ÞC21fðT         1
ð1Þ             1
"coarse-grained"            1
"enter          1
"inner"         1
"to             1
```

```
wðt             2
wðÞ,            2
wðÞC21wðt       2
Þ               1
Þ,              1
Þ;              1
Þg              1
Þg;             1
Þ¼              1
ÞÞg             2
ð2Þ             1
ð3Þ             1
ð4Þ             1
ðc;G;w;TÞ       1
ði;G;w;TÞ       1
þjVjþjEjÞ.      1
"allowable"     2
"greedy"        2
```

Figure 6: Word count processing in Internal cloud



An Efficient Method For Word Count Across
External cloud and Internal cloud

Final Output

```
!               6
(DAG),          1
(DSP).          1
(Dependency     1
(Functional     1
(Independent    1
(Open           1
(Rate           1
(SIM)           1
(Test           1
(The            1
(a              1
(defined        1
(recall         1
(withdraw),     1
,               5
,butnotnecessarilyimmediatelybeforet            1
.               8
0               3
00              2
```

Figure 7: Combined output of EC & IC

Figure 8 gives the comparison analysis of time taken for execution by existing system and proposed system. Here existing system is Task stealing approach, which is compared with proposed system i.e. BStream approach. In task stealing approach are based on previous version of hadoop where there is fixed partitioning of slots into map slots and reduce slots. Here, the reducers start right at the beginning of the computation. YARN a newer version of hadoop is used in Bstream, In YARN this fixed partitioning is no longer maintained. As in below figure execution time taken by existing system is much higher as compared time taken by proposed system.

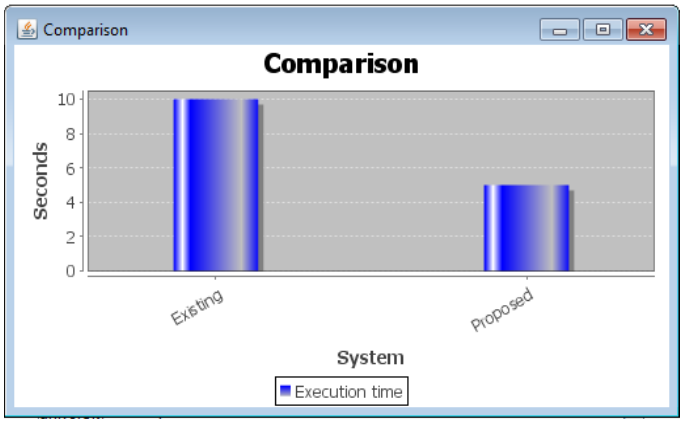


Figure 8: Comparison Chart

## V. CONCLUSION

In this paper, we presented BStream, which extends Hadoop to multiple clouds by combining stream processing in External cloud with batch processing in internal cloud. BStream uses an analytical model to estimate resource allocation and task distribution across clouds to meet deadlines. We showed that the performance of analytical model is reasonably accurate. We compared the performance of BStream with other existing works and showed that stream processing along with continuous checkpointing in external cloud can significantly improve performance. Finally, we characterized the operational regime of BStream, paving way for meeting deadlines with multiple jobs.

## REFERENCES

[1] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "*Starfish: A self-tuning system for big data analytics*," in Proc. 5th Int. Conf. Innovative Data Syst. Res., 2011, pp. 261–272.

[2] C. Vecchiola, R. N. Calheiros, *et al.*,"*Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka*," Future Generation Comput. Syst., vol. 28, no.1, pp. 58–65, 2012.

[3] T. Bicer, D. Chiu, and G. Agrawal,“*Time and cost sensitive data-intensive computing on hybrid clouds,*” in Proc. IEEE/ACM 12th Int. Symp. Cluster, Cloud Grid Comput, 2012, pp. 636–643.
[4] S. Kailasam, N. Gnanasambandam, J. Dharanipragada, and N.Sharma, “*Optimizing ordered throughput using autonomic cloud bursting schedulers,* ”IEEE Trans. Softw. Eng., vol. 39, no. 11, pp. 1564–1581, Nov. 2013.
[5] K. Kc and K. Anyanwu, “*Scheduling hadoop jobs to meet dead-lines,”* in Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci., 2010, pp. 388–392.
[6] M. Mathess and R. N. Calheiros, R. Buyya,“*Scaling MapReduce applications across hybrid clouds to meet soft deadlines*,” in Proc. IEEE 27th Int. Conf. Adv. Inf. Netw. Appl., 2013, pp. 629–636.
[7] Y. Luo, Z. Guo, Y. Sun, B. Plale, J. Qiu, and W. W. Li*,“A hierarchi-cal framework for cross-domain MapReduce execution*,” inProc. 2nd ACM Int.Workshop Emerging Comput. Methods Life Sc., 2011,pp. 15–22.
[8] C. Olston, G. Chiou, L. Chitnis, ***et al.***, “*Nova: continuous Pig/Hadoop workflows,”* in Proc. ACM Int. Conf. Manage. Data, 2011, pp. 1081–1090.