V for Vicissitude: The Challenge of Scaling Complex Big Data Workflows

Bogdan Ghiț, Mihai Capotă, Tim Hegeman, Jan Hidders, Dick Epema, and Alexandru Iosup
Parallel and Distributed Systems Group, Delft University of Technology, The Netherlands
{B.I.Ghit, M.Capota, T.M.Hegeman, A.J.H.Hidders, D.H.J.Epema, A.Iosup}@tudelft.nl

***Abstract*—In this paper we present the scaling of BTWorld, our MapReduce-based approach to observing and analyzing the global BitTorrent network which we have been monitoring for the past 4 years. BTWorld currently provides a comprehensive and complex set of queries implemented in Pig Latin, with data dependencies between them, which translate to several MapReduce jobs that have a heavy-tailed distribution with respect to both execution time and input size characteristics. Processing BitTorrent data in excess of 1 TB with our BTWorld workflow required an in-depth analysis of the entire software stack and the design of a complete optimization cycle. We analyze our system from both theoretical and experimental perspectives and we show how we attained a 15 times larger scale of data processing than our previous results.**

I. Introduction

Global networks with millions of nodes, from the BitTorrent file-sharing network to the smart electricity grids, are increasingly used to support daily human activities, requiring continuous monitoring and analysis. Starting with 2009, in our BTWorld project [1] we have conducted a longitudinal experiment in observing the global BitTorrent network during which we have collected over 15 TB of operational data. Although we have created a MapReduce-based logical workflow to extract insightful knowledge about the evolution of the BitTorrent network [2], the vicissitude of processing our BitTorrent data, that is the combination between large volume of data and the complexity of the processing workflow, has prevented us until now to gather useful insights. To address this problem, in this work we demonstrate the scaling of the BTWorld workflow and process an order of magnitude more data than in our previous attempt [2]. The contributions of this paper are conceptual, technical, and related to scale.

In terms of conceptual contribution, we introduce a complex workflow for big data processing solving a real-world problem that is highly relevant for the evolution of the Internet, which is the analysis of the operation of the global BitTorrent file-sharing network. Popular yet simple benchmarks used in much of the big data research fail to stress the MapReduce software stack. In contrast, the level of complexity of our workflow drives the MapReduce platform to its limits which unexpectedly crashed multiple times. Thus, scaling it to very large data sets required several considerations, both in terms of tuning the platform configuration and workflow design. The BTWorld workflow currently includes 17 high-level SQL-like queries implemented in Pig Latin which translate to 35 MapReduce jobs. The complexity of the workflow turned out to be more important than the data volume. While trying to scale BTWorld to larger data sets, the vicissitude encountered determined us to analyze in-depth the entire software stack and understand the impact of each level on the actual performance of the workflow.

Our technical contributions form a complete optimization cycle that includes modifications applied at all levels of the big data processing stack. At the storage level, we show the importance of input data organization and the influence of the replication factor on the execution time of complex workflows. At the level of MapReduce, we found that delaying the execution of reducers can alleviate memory pressure and that increasing default buffer allocations is essential. At the high-level programming level, Pig, we show that finding a balance between the number of tasks and memory allocation is crucial to avoid crashes. Finally, at the workflow level, it is important for the design to include inter-query data reuse.

As to scale, we have processed 40 times more data compared to the "biggest" presentation at CCGrid SCALE 2013, MR-DBSCAN. At the same time, we surpass our own previous work by a factor of 15 [2]. Most importantly though, the problem we are addressing in this paper concerns the global BitTorrent network, which is of an unprecedented scale in peer-to-peer research. The scientific community can greatly benefit from our analysis to improve the service level of the hundreds of millions of BitTorrent users [3], but also to alleviate the network issues that BitTorrent can cause because of its heavy use of upload bandwidth [4].

In summary, the contributions of this paper are threefold:

1) We present BTWorld, our MapReduce-based approach for observing the global BitTorrent system.
2) We design a multi-level optimization cycle for tuning MapReduce-based software stacks in order to accommodate complex workflows.
3) We present the scaling of the BTWorld workflow to 15 times the size of the data sets of our previous results [2].

II. Observing the Global BitTorrent Network

BitTorrent is the most popular peer-to-peer file sharing protocol and the biggest Internet application in terms of residential upload bandwidth usage [4]. BTWorld [1] is an ongoing project we started in 2009 to observe the global BitTorrent network.

978-1-4799-2784-5/14 $31.00 © 2014 IEEE
DOI 10.1109/CCGrid.2014.97

IEEE computer society

TABLE I
OVERVIEW OF THE COMPLETE BTWORLD DATA SET.

Collection period	2009-12-04 to 2014-02-06
Total size of data set	14.90 TB
Unique swarm samples (estimate)	165 billion
Unique trackers	2 378
Unique timestamps	79 521
Unique scrapes	10 001 940

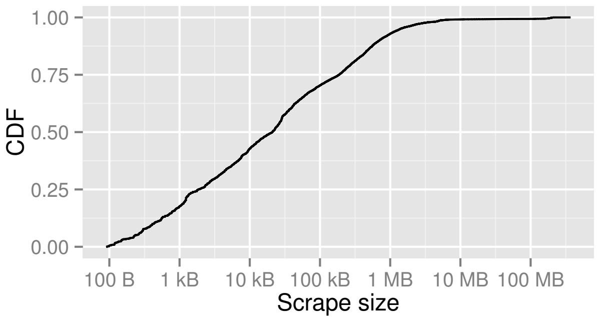


Fig. 1. CDF of mean scrape size per tracker. Logarithmic horizontal scale.

TABLE II
QUERIES OF THE LOGICAL WORKFLOW IN BTWORLD.

Acronym	Query Description
`ToT`	Tracker status over time
`AT` / `AS` / `AH`	Active trackers/swarms/tracker per timestamp
`TKTL` / `TKTG`	Local/global top-K trackers
`TKHL` / `TKHG`	Local/global top-K hashes
`TKSL` / `TKSG`	Local/global top-K swarms
`TKNDH`	Newborn/Dead hashes over time for top-K trackers
`SeT` / `SwT` / `SLT`	Sessions/Swarms/SeederLeecherRatio over time
`TT` / `TTS` / `THS`	Total number of trackers/timestamps/hashes&swarms

A. BitTorrent and BTWorld Operation

Sharing of a file in BitTorrent starts with the creation of a *torrent*–a metadata file–containing information about the file to be shared. The file is logically broken up into constant length pieces which are SHA-1 hashed to facilitate the file transfer–different pieces can be downloaded from different peers in parallel and peers can upload pieces even before having the complete file. Piece hashes are recorded in the torrent, together with the file name, and are themselves hashed to obtain a unique identifier–a *hash*–allowing peers interested in the file to find each other. Centralized servers called *trackers* are used by peers to find members of the *swarm* of peers sharing the file identified by a certain hash.

BTWorld aims to capture the global state of BitTorrent; instead of monitoring the hundreds of millions of peers themselves [3], which would require resources well beyond the possibilities of a university lab, BTWorld contacts trackers to gather statistics. We built a comprehensive list of trackers which we query periodically using several Linux servers running `cron` and `wget`. Each query response, or *scrape*, contains, for each hash at the tracker, the following swarm statistics: the number of *seeders*–peers having the complete file, the number of *leechers*–peers downloading the file, and the total number of downloads. The raw scrape data is encoded in a BitTorrent specific format [5] and can be further gzip-compressed by the tracker for transport over HTTP.

B. Global Scale

The BTWorld project operates on an unprecedented scale in the area of file sharing in terms of population and time. While previous work has focused on at most a handful of trackers to obtain exemplary evidence about BitTorrent phenomena during a few months [6], BTWorld provides researchers a global vantage point from which to observe the evolution of BitTorrent over the past 4 years.

Processing the BTWorld data reveals long term trends like the year-over-year variation in the global peer population, the geographical migration of trackers, or the availability of very old files. At the same time, the data captures the effect of external disruptions on the BitTorrent ecosystem, including legal challenges in several countries [7] and the creation of fake trackers and swarms [1].

C. Data Set

We summarize the characteristics of the data collected by BTWorld from its start in 2009 until now in Table I. The raw tracker scrapes represent almost 15 TB of data. We estimate there are approximately 165 billion unique samples of swarm statistics in the data set based on an average sample size of 90 bytes. BTWorld tracks the evolution of over two thousand trackers from which it has collected more than 10 million scrapes. The distribution of scrape size across all trackers and timestamps is presented in Figure 1. The median is 21 kB, while the mean is 1.49 MB. Most of the scrapes are small, but some trackers generate more than 300 MB of data per timestamp. This skewness makes the data especially hard to process.

III. BTWORLD WORKFLOW

Given the richness of the data set, it is common for the BitTorrent analyst to design new queries, which could (and in our experience do) traverse the entire data set to produce their output. At the same time, queries need to be translated to executable code as fast and simply as possible.

The BTWorld workflow is designed as a set of inter-dependent queries written in Pig Latin, an SQL-like programming language designed to be automatically translated to MapReduce jobs. The queries are presented in Figure 2, together with their data dependencies created to maximize data reuse. Table II summarizes their function (the last six queries in the table are omitted from the figure for clarity). In the rest of this section, we describe three queries that are representative for the diversity of the workflow.

TrackerOverTime (ToT). *What is the evolution of individual trackers?* This query is used to extract information about the evolution of each tracker we monitor in terms of

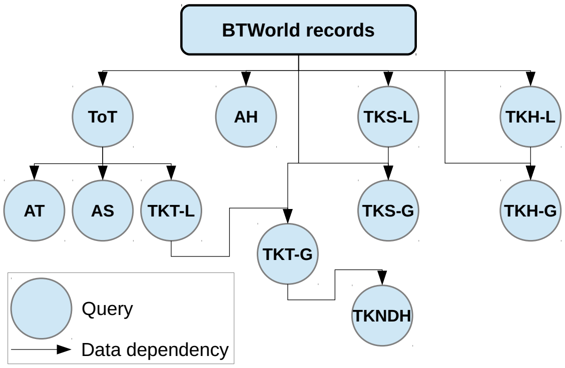


Fig. 2. The BTWorld logical workflow diagram.

Listing 1. Pseudo-code for the ToT query.

```
SELECT tracker, timestamp,
  COUNT(hash) AS hashcount,
  SUM(seeders + leechers) AS sessions,
  AVG(leechers == 0 ?
      seeders : seeders / leechers)
      AS slratio
FROM logs
GROUP BY tracker, timestamp;
```

the number of swarms served, the total peer population (total sum of seeders and leechers), as well as the service level in its swarms (using the ratio of seeders to leechers as a proxy). Listing 1 shows the query. The complete data set is grouped by tracker and timestamp and aggregation functions are applied to extract the statistics for each group.

As a sample of BTWorld output, we plot the CDF of the total number of sessions per tracker as produced by the ToT query on our 1.5 TB data set in Figure 3. The data was sanitized similarly to the original BTWorld paper [1].

ActiveSwarms (AS). *What is the total number of swarms active in the system?* Starting with the output of the ToT query instead of the complete data set considerably speeds-up the execution of this query. This type of inter-query data dependency is used whenever possible throughout the workflow. The pseudocode is shown in Listing 2. The data is grouped by timestamp and the SUM function is applied to the hashcount data pre-computed by the ToT query to extract the number of swarms active in the global network at each timestamp.

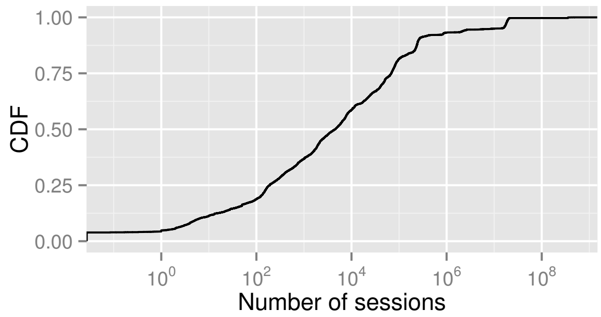


Fig. 3. CDF of the total number of sessions per tracker for all scrapes in the 1.5 TB data set. Logarithmic horizontal axis.

Listing 2. Pseudo-code for the AS query.

```
SELECT timestamp,
  SUM(hashcount) AS swarms
FROM ToT
GROUP BY timestamp;
```

Top-K-Swarms-Per-Time (TKSL). *Which are the K most popular swarms?* To answer this question we designed a complex query that requires the entire data set and several processing stages. Firstly, we group the input data set by hash, tracker and a configurable time window representing the period we investigate and we calculate for each such group the average number of sessions. Secondly, we re-group the data by tracker and time window and select from each resulting group the largest K hashes with respect to the average number of sessions.

IV. Experimental Results

In this section we present the design of the experimental setup we employed to execute the BTWorld workflow and we analyze its performance on a real-world cluster system.

A. System Configuration

We run experiments on the Dutch six-cluster wide-area computer system DAS-4 [8] with node configuration summarized in Table III. The TU Delft cluster, which we use for our big data challenge, has 24 dual-quad-core compute nodes with 24 GiB memory per node and 50 TB total storage, connected within the clusters through 1 Gbit/s Ethernet (GbE) and 20 Gbit/s QDR InfiniBand (IB) networks. In our experiments, we use a standard setup of Hadoop-1.0.0 over InfiniBand. We configure the HDFS on a virtual disk device (with RAID 0 software) that runs over 2 physical devices with 2 TB storage in total per node.

TABLE III
Node Configuration

Processor	Dual quad-core Intel E5620
Physical cores	8
Memory	24 GiB RAM
Physical Disk	2 WD Re 1 TB (RAID 0)
Network	20 Gbit/s InfiniBand
Operating system	CentOS 6 (Linux 2.6.32)
JVM	Oracle JDK 1.6.0_27

B. Optimization Cycle

Scaling the BTWorld workflow for large data sets is not trivial. We have encountered unexpected crashes due to insufficient hardware capabilities (disk quotas and memory capacity exceeded). Using common rule-of-thumb settings of

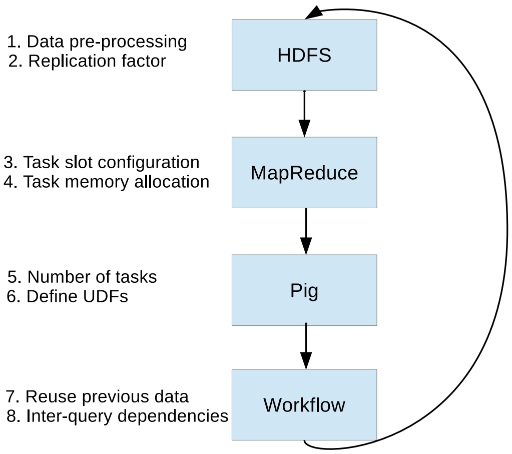


Fig. 4. Different aspects addressed in the software stack and the optimization cycle required by BTWorld.

MapReduce for configuring the number of map and reduce slots, number of reducers, or the replication factor proved to be inefficient for the complex and diverse BTWorld jobs. Thus, to successfully scale the workflow to larger input data sizes, we apply the optimization cycle in Figure 4, which we further discuss in this section.

HDFS. The BTWorld raw input data is decompressed and decoded from the BitTorrent-specific encoding to plain-text files containing tab-separated values, using a combination of shell and Python scripts. Upon insertion into HDFS, the files are organized in a tracker-based directory structure. Scrape samples are concatenated into fixed-size files configured to match the HDFS block size, 128 MiB. Because the largest queries are disk-intensive and generate outputs comparable in size with the input data, we decrease the replication factor in HDFS from the default value 3 to 2 replicas (minimum for fault tolerance). This also accelerates execution time, because our inter-query dependencies imply writing data to HDFS in between MapReduce jobs and this writing also benefits from the reduced replication. As the FIFO scheduler does not optimize for task and data co-location [9], there is no concern about decreasing the data locality when lowering the replication factor.

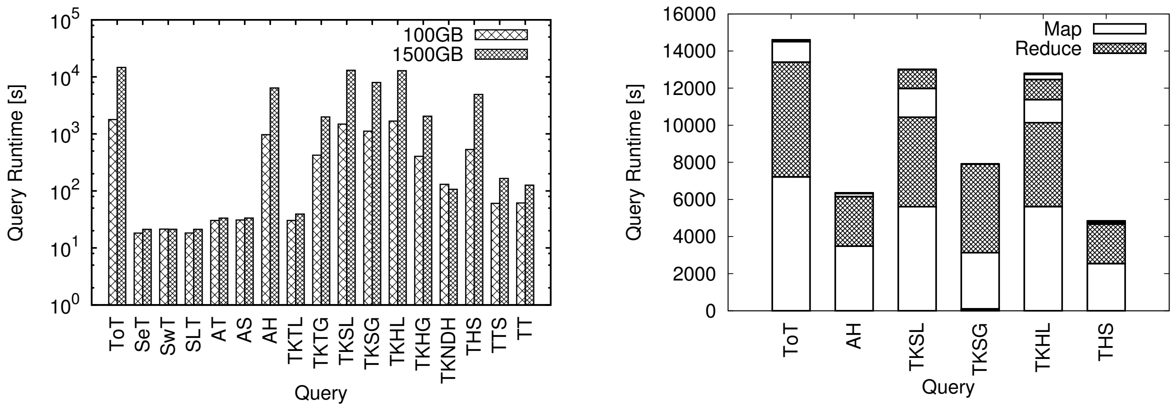
TABLE IV
HADOOP CONFIGURATION

Hadoop version	1.0.0
Cluster size	24
Scheduler	FIFO
Map slots	92
Reduce slots	92
Memory per task	6 GiB
HDFS replication	2

MapReduce. MapReduce scheduler manages internally an overlapping queue which splits the job into map and reduce tasks such that different types of tasks may run simultaneously. The scheduler first executes the map tasks of the jobs, but the overlapping model allows the scheduler to execute reduce tasks when a certain fraction of map tasks have been successfully completed. The general assumption of allowing map and reduce tasks co-exist is that the latter type of tasks only transfer output data from the map tasks, thus not increasing the resource contention of the cluster nodes. Although this is a common configuration for most of the MapReduce traditional applications (e.g., cite HiBench, Facebook), a high degree of task-level concurrency may break the system when running memory-bound applications. We have encountered this issue with our largest BTWorld queries (e.g., ActiveHashes, TopKSwarmsPerTime, TopKHashesPerTime). In consequence, we limited the number of concurrent tasks at 4 each with 6 GiB memory allocated by setting both map and reduce slots set to 4 and stalling the execution of the reduce tasks until all map tasks have completed.

Pig. Although Pig language provides a limited level of control on the optimization of the MapReduce jobs generated, we observed two important elements that influence the implementation and the performance of the workflow: the lack of a wide range of operators and the number of reducers launched by the MapReduce jobs. For the former, we implemented several user-defined functions to support different operations on the timestamps. Given the diversity and the number of the jobs in our workflow, setting the number of reducers to a static value is sub-optimal. However, as Pig allows configuring the number of reducers per query and in our implementation each query may be translated into several jobs, we were not able to tune the number of reducers per job. Therefore, we applied the default rule-of-thumb and set the number of reducers proportionally to the input data of the job (1 GB per reduce task).

BTWorld. The design and implementation of the workflow itself went through several iterations. Given the large size of data generated by the queries, an important aspect was to make efficient use of the data from previously executed queries. One technique used to increase data reuse is identifying common execution patterns and splitting them off into separate queries. For example, both the number of trackers and swarms per timestamp can be found by grouping by tracker and timestamp. To prevent executing this operation twice, we created the TrackerOverTime (ToT) query to perform the grouping and aggregation once, and extract the number of trackers and swarms from the result. A second technique involves rewriting queries to use output of existing queries. For example, BTWorld extracts information about both the global (all-time) and local (per-timestamp) top-K swarms. Intuitively, the global top-K can be found by grouping the full data set by swarm and computing the size of each. However, as the size of a swarm is defined as its peak concurrent sessions, the size of a swarm can be extracted from the local top-K results (a swarm in the global top-K must be part of the local top-K during its peak).



(a)

(b)

Fig. 5. The performance of the BTWorld queries. The query runtimes compared for 100 GB vs. 1500 GB (a), in logarithmic vertical scale, and the query runtimes of the largest queries decomposed in successive map-reduce phases (b).

TABLE V
EXPERIMENT OVERVIEW

Data set size	1500 GB	100 GB
Pig queries	17	17
MapReduce jobs	35	35
Total map tasks	101 955	6987
Total reduce tasks	7721	745
Total runtime	17.87 h	2.44 h
Shuffled data (temporary)	11.9 TB	1.4 TB
Intermediary data (persistent)	1.7 TB	73 GB

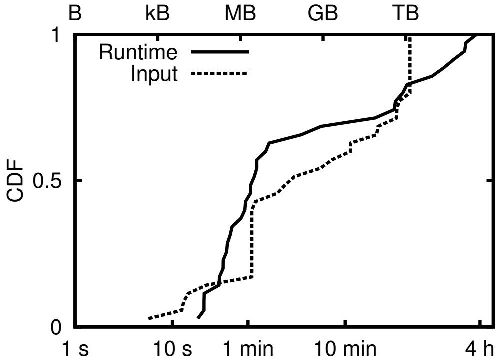


Fig. 6. The BTWorld job runtime and input data size distributions for 1500 GB. Logarithmic horizontal scale.

C. BTWorld Performance

In this section, we extract the characteristics of the workflow after running BTWorld on a 24-node Hadoop cluster deployed on our DAS-4 system with the configuration given in Table IV. We compare the query performance for two data sets, and we show the execution time across different MapReduce phases for a selection of the largest queries.

The general characteristics of the workflow are depicted in Table V. The 35 MapReduce jobs launch over 110 000 map and reduce tasks for the 1.5 TB data set which are executed in almost 18 hours. The amount of shuffled data between the map and reduce phases is more than 10 times larger than the initial input. Moreover, the size of the intermediary data resulted from our design of the BTWorld workflow has the same order of magnitude as the input. While the former is discarded when the jobs are completed, the latter is persistent and possibly used by other jobs (e.g., the output data of ToT is later used by AS and AT, see Figure 2).

Most of the queries (e.g., SeT, SwT, SLT) run over summaries of the entire data set, whose sizes are quasi-independent of the original input. Thus, as we illustrate in Figure 5(a), their performance is comparable even when the input data size is increased by a factor of 15, switching from the 100 GB to the 1.5 TB data set.

A small fraction of the BTWorld queries (i.e., ToT, AH, TKSL, TKSG, TKHL, and THS) determine the performance of the entire workflow, because they process the entire data set. These queries are responsible for almost half of the total load of the system and translate into multiple MapReduce jobs, with non-overlapping map and reduce phases (see Figure 5(b)).

In Figure 6, we depict the distributions for the job runtime and input size for the complete BTWorld workflow, consisting of 35 jobs. The mean and median runtimes are 1838 s and 38.69 s, respectively; the mean and median input data sizes are 373 MB and 42 MB, respectively. Similarly to workloads in production clusters from Google and Facebook, the jobs in the BTWorld workflow follow a heavy-tailed distribution with respect to both runtime and input data size.

V. CONCLUSIONS

Overall, the most important lesson learnt from implementing the BTWorld workflow is that the promise of high-level languages like Pig of making the big data processing stack available to everybody, without requiring expertise in the lower levels of the stack, is not fulfilled. The performance of MapReduce workloads is dependent on several aspects such as the data set layout and size, the hardware configuration of the cluster, or the deployment of the MapReduce framework within the physical infrastructure. Current techniques for tunning the performance of MapReduce workloads, such as Starfish [10], consider these elements as independent variables, without studying how they influence each other. Instead, based on our experience with BTWorld, we argue that finding the correlations between different parameters is a building block towards an optimized data processing framework.

The design space for autonomous systems is still large. We experienced crashes at all levels, even for simple queries, and scaling the system to process data sets in excess of one terabyte required a wide range of manual tuning. Nevertheless, autotuning techniques based on machine-learning algorithms [11] are offline, which makes them impractical for multiple reasons. Training the models requires repeated runs, which is time-consuming even for single simple jobs such as Wordcount or Sort. This issue is amplified for long running jobs or complex workflows like ours. Furthermore, as the optimal configuration depends on the hardware configuration of the cluster, the training phase has to be re-executed when moving from one infrastructure to another. Therefore, we argue the need of an online performance tunning tool for data analytics, which can explore the parameter space during the system operation and eventually converge to the optimal offline configuration.

Although designed for dedicated single cluster deployments, data processing platforms like MapReduce can benefit from elastic scaling [12]. Based on the observations we have made in the current configuration of our system, we also identify the need to scale out the BTWorld processing. We are currently working on a multi-cluster deployment of the MapReduce framework which will enable a larger capacity for running the BTWorld workflow.

Finally, we have learnt that benchmarking big data processing systems using common yet simple workload suites (e.g., [13],[14]) with small data volumes may not be representative for an accurate performance evaluation. Instead, we advocate using more complex applications in current big data research.

VI. Acknowledgement

This publication was supported by the Dutch national program COMMIT and the STW/NWO Veni grant 11881.

References

[1] M. Wojciechowski, M. Capotă, J. Pouwelse, and A. Iosup, “BTWorld: Towards Observing the Global BitTorrent File-Sharing Network,” *LSAP Workshop in conjunction with HPDC*, 2010.

[2] T. Hegeman, B. Ghiț, M. Capotă, J. Hidders, D. H. J. Epema, and A. Iosup, “The BTWorld use case for big data analytics: Description, MapReduce logical workflow, and empirical evaluation,” *2013 Int’l Conf. on Big Data*. IEEE, Oct. 2013, pp. 622–630. [Online]. Available: http://dx.doi.org/10.1109/BigData.2013.6691631

[3] BitTorrent, Inc., “BitTorrent and Torrent Software Surpass 150 Million User Milestone.” [Online]. Available: http://www.bittorrent.com/company/about/ces_2012_150m_users

[4] Sandvine, “Global Internet Phenomena Report 1H2013.”

[5] B. Cohen, “The BitTorrent Protocol Specification.” [Online]. Available: http://bittorrent.org/beps/bep_0003.html

[6] C. Zhang, P. Dhungel, D. Wu, and K. W. Ross, “Unraveling the bittorrent ecosystem,” *IEEE TPDS*, Vol. 22, no. 7, pp. 1164–1177, 2011.

[7] J. Poort, J. Leenheer, J. van der Ham, and C. Dumitru, “Baywatch: Two Approaches to Measure the Effects of Blocking Access to the Pirate Bay,” *SSRN Electronic Journal*, 2013. [Online]. Available: http://dx.doi.org/10.2139/ssrn.2314297

[8] www.cs.vu.nl/das4/.

[9] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling,” *EuroSys*, 2010.

[10] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, and S. Babu, “Starfish: A Self-tuning System for Big Data Analytics,” *5th Biennal Conference on Innovative Data Systems Research (CIDR)*, 2011.

[11] N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema, “Towards machine learning-based auto-tuning of mapreduce,” *21st MASCOTS*. IEEE Computer Society, 2013, pp. 11–20.

[12] B. Ghit, N. Yigitbasi, and D. Epema, “Resource Management for Dynamic MapReduce Clusters in Multicluster Systems,” *High Performance Computing, Networking, Storage and Analysis (SCC), SC Companion*. IEEE, 2012, pp. 1252–1259.

[13] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, “The Hibench Benchmark Suite: Characterization of the MapReduce-based Data Analysis,” *ICDEW*, 2010, pp. 41–51.

[14] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, “The Case for Evaluating MapReduce Performance Using Workload Suites,” *MASCOTS*, 2011, pp. 390–399.