Delft University of Technology
Parallel and Distributed Systems Report Series

The BTWorld Use Case for Big Data Analytics: Description, MapReduce Logical Workflow, and Empirical Evaluation

Tim Hegeman, Bogdan Ghiț, Mihai Capotă, Jan Hidders, Dick Epema, and Alexandru Iosup
T.M.Hegeman@student.tudelft.nl

Completed October 2013.

Report number PDS-2013-008

ISSN 1387-2109

Published and produced by:
Parallel and Distributed Systems Group
Department of Software and Computer Technology
Faculty Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
reports@pds.ewi.tudelft.nl

Information about Parallel and Distributed Systems Section:
http://www.pds.ewi.tudelft.nl/

Abstract

The commoditization of big data analytics, that is, the deployment, tuning, and future development of big data processing platforms such as MapReduce, relies on a thorough understanding of relevant use cases and workloads. In this work we propose BTWorld, a use case for *time-based big data analytics* that is representative for processing data collected periodically from a global-scale distributed system. BTWorld enables a data-driven approach to understanding the evolution of BitTorrent, a global file-sharing network that has over 100 million users and accounts for a third of today's upstream traffic. We describe for this use case the analyst questions and the structure of a multi-terabyte data set. We design a MapReduce-based logical workflow, which includes three levels of data dependency—inter-query, inter-job, and intra-job—and a query diversity that make the BTWorld use case challenging for today's big data processing tools; the workflow can be instantiated in various ways in the MapReduce stack. Last, we instantiate this complex workflow using Pig–Hadoop–HDFS and evaluate the use case empirically. Our MapReduce use case has challenging features: small (kilobytes) to large (250 MB) data sizes per observed item, excellent (10^{-6}) and very poor (10^{2}) selectivity, and short (seconds) to long (hours) job duration.

Contents

List of Figures

List of Tables

1 Introduction

Time-based analytics, that is, extracting meaningful information out of a very large set of timestamped information, is challenging for existing data processing systems such as the popular MapReduce-based Hadoop [1, 2, 3], which must be operated efficiently to achieve good performance cheaply [4]. Understanding the workload, through use cases or real-workload traces, can significantly help tune existing systems [5, 6] and improve future-system designs [7]. In this work, we present the BTWorld [8] use case for time-based big data analytics, which aims at understanding the recent evolution of BitTorrent, a major Internet application with significant traffic and over 100 million users. Our use case extends prior work on MapReduce workloads with a comprehensive use case that focuses on a new application domain, increased diversity of analytics, a workflow of coupled MapReduce jobs, and an empirical study based on a multi-year data set. With BTWorld, we are also able to extend over a decade of theoretical BitTorrent research with knowledge that can only be acquired from a big-data-driven study. We further discuss the idea of extending the BTWorld use case towards a benchmark suite for time-based analytics platforms.

Time-based analytics can lead to knowledge otherwise inaccessible to analysts, but pose interesting new challenges to big data processing systems. Large amounts of time-stamped new records are added periodically to a continuously growing data set, creating time series of various lengths. Studying the evolution in time of an observed system that may include tens of millions of objects, as the use case introduced in Section 2 does, may involve algorithms for log processing that have variable complexity based on both the amount of the data that needs to be mined and the operations performed on the data. The analyst may ask for various operations, including through SQL-like aggregations, selections, joins, or projections. The analyst may also formulate complex queries that involve large amounts of or even the entire data set, such as "What is the evolution of the most relevant objects in the system, over the entire measurement?". Minimizing the amount of the data processed by each query, understanding and exploiting data reuse, selecting the order of execution for the queries, and many other approaches must be designed, automated, and tuned for efficiently managing the query workflow.

Designing, automating, and tuning data processing systems all rely on a good understanding of the workload, and in particular of three main types of data dependency. *Inter-query dependencies* occur when the output of a query can be reused as input by another. For example, the most popular K items sold by a chain of stores can be extracted from the output of a query that computes the most popular K items for each store. *Inter-job dependencies* occur when complex queries with different operators are translated into workflows of jobs, such that jobs cannot start before the output of their predecessors is materialized. A common example of inter-job data dependency is given by the TeraSort benchmark, which includes three jobs, for data generation, actual sorting, and output validation. *Intra-job dependencies* occur when data-intensive frameworks exploit the available job parallelism by breaking down each job into multiple tasks. In many popular programming models, from the 1990s BSP [9] to the modern MapReduce, a job may have one or even multiple synchronization points which split(s) the computation into multiple phases with identical tasks within each phase, but (possibly) distinct tasks between different phases. For MapReduce, there are two main phases: the map phase which performs a group by on each partition of the data set, followed by the reduce phase which aggregates the output of the map phase.

We focus in this work on the BTWorld use case, as an example of big data time-based analytics. In particular, we focus on a MapReduce-based workflow and implementation, with a broader discussion towards a benchmark for time-based analytics platforms. The MapReduce programming model has caught the attention of many scientific and engineering laboratories

around the world, with over 10 000 distinct programs implemented only at Google [10], thousands of updates provided by the leading providers of commercial MapReduce software stacks to the open-source Hadoop, and hundreds of thousands of processors managed with the Hadoop main middleware [11]. Prior work has already focused on understanding MapReduce workloads [12, 13, 12], presented several MapReduce use-cases [14, 7], or focused on MapReduce benchmarking suites [15, 16, 17]. In contrast, our work focuses on a new application domain, a large input data set, and a real and complex data processing workflow with diverse queries. Our main contribution is three-fold:

1. We describe a use case for big data, time-based analytics (Section 2). Our use case, BTWorld [8], represents the observation of massive, complex, distributed systems operated by millions of users—the global BitTorrent file-sharing network. We present the use case from a big data perspective, focusing on a multi-year, multi-terabyte data set and on a set of questions that the system analyst asks the data processing system to answer. This use case can be very useful for the many domains that increasingly study large complex systems through data-driven approaches: business intelligence and decision making, computer science, social sciences, urban planning, etc.

2. We design a MapReduce-based logical workflow that can answer the set of questions that the system analyst asks (Section 3). Our logical workflow, for which queries exhibit all three levels of data dependency, extends the current body of work in MapReduce use cases and benchmarks. In practice, the workflow can be implemented using various open-source and commercial tools from the MapReduce stack, and thus enables their comparison for a relevant application domain.

3. We implement the logical workflow using the Pig Latin–Hadoop–HDFS MapReduce stack and conduct an empirical evaluation with subsets of increasing size extracted from the complete BTWorld data set (Section 4). We analyze system-level, query-level, and MapReduce job- and task-level results. We show evidence of the diversity of the queries proposed, in terms of execution time and resource consumption (including I/O).

2 Description of the BTWorld Use Case

In this section, we describe the BTWorld use case from a big data perspective. Started in 2009 and ongoing, the BTWorld project [8] aims to measure the global BitTorrent network, which is the largest Internet application in terms of upstream traffic (generating 35% of global upstream traffic in 2013 [18]). BTWorld collects periodically snapshots of the global BitTorrent network, which it aims to use to answer many types of research questions. We focus in this section, in turn, on BTWorld background and terminology, on a multi-terabyte BTWorld data set, and on exemplary research questions BTWorld aims to answer. Outside the scope of this work, we are gaining through BTWorld unique insights into the evolution of BitTorrent over the past 3.5 years, extending and complementing over a decade of theoretical work.

2.1 BitTorrent and BTWorld Overview

BitTorrent is a peer-to-peer (P2P) file-sharing protocol whose success comes mainly from facilitating and incentivizing collaboration between peers. BitTorrent breaks up files into SHA-1 hashed pieces that can be shared individually by peers, even by peers who do not possess the complete file. For each file shared in BitTorrent, the file name and the hashes of its pieces form a metadata file (a *torrent*), which is uniquely identified by a SHA-1 hash of the piece hashes and

Table 1: Overview of the complete BTWorld data set.

Collection period	2009-12-04 to 2013-06-17
Total size of data set	14.42 TB
Unique swarm samples (estimate)	150 billion
Unique trackers	2 369
Unique timestamps	70 443
Unique scrapes	8 422 938

file name. A *swarm* is a group of BitTorrent peers sharing the same torrent. Among the peers of a swarm, *seeders* posses all the pieces, while *leechers* possess only some of the pieces and are downloading the remainder. To help peers meet each other, for example to join a swarm for the first time, BitTorrent also uses *trackers*, which are centralized servers that give upon request lists of peers in the swarm of a particular torrent. Through this mechanism, different trackers can generate different swarms for the same torrent.

BTWorld focuses on understanding BitTorrent and its evolution, which have a significant impact in the operation of the entire Internet. Traditional BitTorrent theory, such as the fluid model of Qiu and Srikant [19], can predict interesting steady-state phenomena, but fails to account for complex transient behavior (e.g., flashcrowds); for complex technical limitations (e.g., firewalls); for complex inter-dependencies between global BitTorrent elements (e.g., legal and auto-feed effects); etc. As a consequence, many important questions related to non-functional system properties—availability, performance, etc.—cannot be answered. As an alternative, with BTWorld we propose a data-driven approach to acquiring knowledge about BitTorrent and perhaps even general distributed systems. By collecting data that can be used in statistical models, machine learning, and validation of theories, BTWorld promises to solve many of the problems faced by the current theoretical approaches and to lead to new theories. However, a data-driven approach raises many challenges in building an efficient, scalable, and cost-effective system for data processing and preservation.

2.2 Data Collection

Studying P2P networks is difficult, as it normally involves monitoring millions of non-cooperating computers. Instead, BTWorld focuses on collecting data from the public trackers of the global BitTorrent network. BTWorld sends queries to (*scrapes*) each tracker and receives statistics about the aggregated status of peers: for each swarm of the tracker, the number of leechers, the number of seeders, and the total number of downloads since the creation of the torrent.

The data is collected by several Linux servers using *wget*, which is run at regular intervals by *cron*. The raw data is bencoded [20] scrape data, either plain text or gzip compressed, depending on the tracker. The data retrieved through *wget* is compressed and stored in a date-based directory structure. Table 1 presents an overview of the data set collected by BTWorld since the project started in 2009. The total size of files amounts to more than 14 TB. Figure 1 shows a CDF of the mean scrape size for all trackers in the data set. The distribution is skewed: the median scrape size is 23 kB, but the largest 1% of the trackers return scrapes sized 200–250 MB. Based on the observed mean sample size of 90 bytes, we estimate the BTWorld data set at approximately 150 billion swarm samples. For Section 4, we use samples of the BTWorld data set ranging from 10 MB to 100 GB to avoid excessive experiment durations.

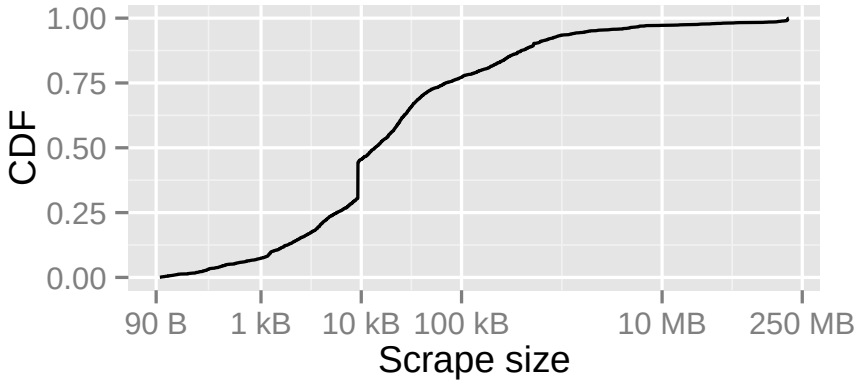


Figure 1: CDF of mean scrape size per tracker.

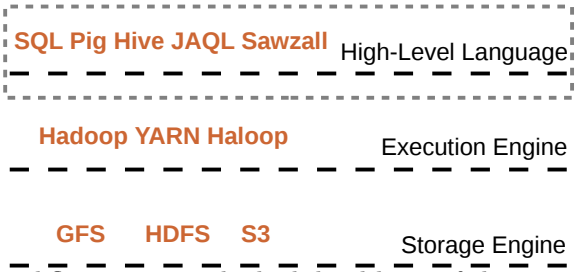


Figure 2: Our logical workflow occupies the high-level layer of the generic MapReduce software stack for data processing.

2.3 P2P Analyst Questions

The data collected by BTWorld during its more than 3.5 years of operation, representing one of the largest longitudinal studies of P2P systems, can be used to answer several questions of interest to peer-to-peer (P2P) analysts [21, 8].

BTWorld can shed light on the evolution of BitTorrent. It captures phenomena such as the seasonal variety in peer population and the shift in the geographical location of the major trackers. It can also show how BitTorrent usage changes. Are the swarms becoming bigger? Is the number of peers per tracker increasing?

The P2P analyst can extract information about the service level provided by BitTorrent to its users by examining the ratio between seeders and leechers in swarms, which is known to be correlated with download speed [22] and with severe degradation of performance during flashcrowds [23]. The life time of swarms is an indicator of reliability [24]: for how long are files available in the system? What is the redundancy level of the system? Is there an overlap between trackers? Are the same hashes present at multiple trackers?

Furthermore, the BTWorld data set contains information about the effect of legal and technical disruptions on the BitTorrent ecosystems [25]. It can show the decline of The Pirate Bay as the leading tracker as a result of a lawsuit against its operators and its replacement by OpenBitTorrent. It also documents the existence of malicious spam trackers designed to impede BitTorrent operation [8].

3 MapReduce-Based Logical Workflow

In this section we design a MapReduce-based data processing approach that can be used to answer the P2P analyst questions introduced in Section 2.3. Our design relies on the MapReduce stack (explained in Section 3.1), which limits applicability to the MapReduce ecosystem, but ensures that our approach can readily be implemented using a variety of open-source and commercial tools. The approach we propose is to implement a logical workflow in a high-level language that can be mapped automatically to the MapReduce programming model and to a MapReduce execution engine. The logical workflow, which does not rely on any practical tool in the MapReduce stack, consists of a data layout (Section 3.2) and a workflow frame that coordinates the execution of several SQL-like queries (Section 3.3). The MapReduce-friendly data layout and the diverse set of queries make the BTWorld logical workflow useful as a MapReduce use case.

3.1 MapReduce Stack Overview

Recently, a general *MapReduce stack* has emerged as a convenient structure for the diverse ecosystem of open-source and commercial middleware that currently support the MapReduce programming model [1]. Figure 2 depicts the three main layers of the MapReduce stack. The *high-level language* layer includes a variety of data manipulation languages and allows analysts to interface with the MapReduce programming model through the language of choice; for example, Pig Latin [26, 27] is an SQL-like high-level language that compiles automatically into MapReduce jobs. The *execution engine* layer implements the MapReduce programming model and typically provides the automatic, reliable, efficient use of computational resources; Hadoop [2] and YARN are open-source execution engines. The *storage engine* layer provides similarly MapReduce-friendly services for data storage and possibly preservation; the Hadoop Distributed File System (HDFS [3]) is a storage engine.

The MapReduce stack is widely deployed and actively maintained. Companies such as Hortonworks and Cloudera, whose commercial platforms integrate all three layers of the MapReduce stack, have provided thousands of updates to the open-source Apache Hadoop in the past five years. The MapReduce stack is typically deployed over a large distributed system (e.g., cluster, grid, cloud), but middleware that can use resources from parallel supercomputers, GPUs, and exotic architectures is also under active development.

3.2 Data Set Layout

We design in this section a MapReduce-friendly data layout. The raw tracker scrape data collected by BTWorld must be introduced into the storage engine, ready for use by the execution engine. The data layout design allows an implementation in several of the open-source storage engines, e.g., HDFS.

First, in our design the raw tracker scrape data collected by BTWorld is decompressed and decoded prior to insertion into the storage engine. This produces tab-separated plain text files, one per tracker and sample. The total size of the files ranges from a few kilobytes for niche trackers with only a few torrents to tens of gigabytes for the biggest trackers with millions of torrents. The records in these files are represented by tuples with the following six fields:

- *Hash (H)*: a SHA-1 hash that uniquely identifies the content transferred in BitTorrent. Represented as a 40-character string of hexadecimal digits.
- *Tracker (TR)*: an URL identifying the BitTorrent tracker.

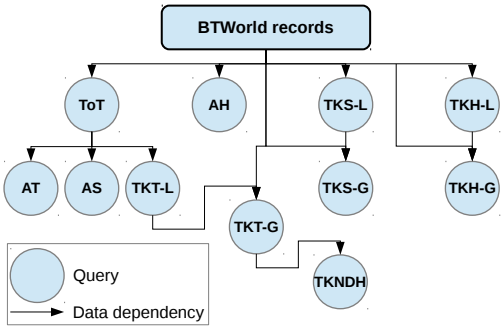


Figure 3: The BTWorld logical workflow diagram. A data dependency is a form of inter-query dependency.

Table 2: Queries of the logical workflow in BTWorld.

Acronym	Query Description
`ToT`	Tracker status over time
`SeT` / `SwT` / `SLT`	Sessions/Swarms/SeederLeecherRatio 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

- *Timestamp (TS)*: the time when the status information was logged. Represented as a 11-character ISO 8601 basic format combined date and time representation.
- *Seeders (S)*: the number of seeders in the swarm at the moment when the sample is taken.
- *Leechers (L)*: the number of leechers in the swarm at the moment when the sample is taken.
- *Downloads (D)*: the number of downloads up to the moment when the sample is taken.

Second, the files are inserted into the storage engine using a tracker-based directory structure. Small scrape files for the same tracker and consecutive timestamps are concatenated to reduce the amount of internal fragmentation.

3.3 Workflow of SQL-Like Queries

We design the BTWorld queries as SQL-like queries integrated into the logical workflow. Similarly to the data layout, the SQL-like queries can be implemented using several tools from the MapReduce stack, e.g., the Pig Latin high-level language or even Hadoop (through manual conversion into the MapReduce programming model).

The logical workflow includes several inter-query dependencies (see Section 1). Figure 3 presents an overview of the full logical workflow for the BTWorld use case. Each node represents an SQL-like query; a query cannot start before the data outputted by each predecessor query in the workflow has been produced. Table 2 summarizes the acronym and meaning of each query. As the workflow diagram and the table indicate, the queries are not one-to-one conversions of every

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;
```

Listing 2: Pseudo-code for the `AS` query.

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

individual analyst question. Instead, to increase performance we have designed our workflow to reuse results (intermediary output data) wherever possible (see Appendix A).

We describe in the following four representative SQL-like queries of the logical workflow. As we will show in Section 4.3, they contain various inter-job and intra-job dependencies (see Section 1) when implemented and exhibit various operational behavior when processing BTWorld data.

TrackerOverTime (ToT). *How does a tracker evolve in time?* We designed this query to monitor the status of a tracker in time with respect to the number of hashes, the number of session (the sum between the seeders and leechers), and the ratio of seeders to leechers. The query, shown in pseudo-code in Listing 1, first groups the input data set by the `key=(TR, TS)`, and then applies different aggregation functions (e.g., count, sum, avg) on the `value=(H, S, L, D)` fields.

ActiveSwarms (AS). *How many active hashes are in the system at every moment of time?* The output of the `ToT` query can be further used to extract the number of active swarms at any given time. The query (Listing 2 implements the same operators as the `ToT` query (group and aggregate). However, the `AS` query runs on a much smaller dataset, and is expected to have different performance characteristics.

ActiveHashes (AH). *How many active hashes are in the system at every moment of time?* Despite the similarity with the `AS` query, the definition of the `AH` query (Listing 3) differs greatly. Because the output of `ToT` cannot be used to count the number of active hashes, the full dataset is processed again. The data is grouped by timestamp and the count-distinct operation is performed on each group to determine the number of unique hashes at each point in time.

Top-K-Trackers (TKT). *Which are the most popular K trackers?* To answer this question, we process the output of the `ToT` query in multiple successive stages. First, we extract the top K trackers per timestamp (the local `TKT` query). Second, we use these results to get the global top K and extract all data for these trackers from the initial data set. Listing 4 presents the pseudo-code for the global `TKT` query.

Listing 3: Pseudo-code for the `AH` query.

```
SELECT timestamp, COUNT(DISTINCT(hash))
FROM logs
GROUP BY timestamp;
```

Listing 4: Pseudo-code for the `TKTG` query.

```
SELECT *
FROM logs
NATURAL JOIN (
  SELECT tracker
  FROM TKTL
  GROUP BY tracker
  ORDER BY MAX(sessions) DESC
  LIMIT k);
```

4 Empirical Evaluation

In this section, we present an empirical performance evaluation of the BTWorld use case. Overall, our results indicate the complexity of the BTWorld use case and that processing the entire BTWorld data set (over 14 TB, as described in Section 2.2) would take about 3 months with our current infrastructure.

We have implemented the MapReduce-based logical workflow in the MapReduce stack through a set of open-source tools, and executed it on subsets of increasing size of the complete BTWorld data set. We describe in Section 4.1 the cluster and software stack, and the workload and workload-related configuration used in our experiments.

For the performance characterization, we present system results, including workflow execution time, system throughput, global I/O utilization, and other resource consumption (all in Section 4.2); an analysis of the Pig queries (Section 4.3); and an analysis of MapReduce jobs (Section 4.4).

4.1 Experimental Setup

We implement the BTWorld use case using the following MapReduce stack: the Hadoop distributed file system (HDFS) as storage engine, Hadoop as execution engine, and Pig Latin [26, 27] as the high-level language. HDFS and Hadoop are popular MapReduce middleware. Pig Latin is one of several open-source, MapReduce high-level languages that offer an SQL-like language and can compile code to MapReduce jobs.

We deploy our MapReduce stack on a dedicated 8-node MapReduce cluster, with the hardware and software configurations summarized in Table 3. Each of the 7 worker nodes runs two map slots and a single reduce slot. The data is stored in blocks of 64 MB. The circular buffer that contains the output generated by a map task is set to 100 MB. The content of the buffer is spilled on disk when the buffer usage reaches 80%. Each (Pig Latin) query uses one reduce task per gigabyte of input data. The K parameter in the Top-K queries is set to 10 for trackers, and 10 000 for swarms and hashes.

We define several performance metrics. The makespan of the workflow is defined as the

Table 3: Configuration of MapReduce-cluster nodes.

Processor	Dual-core 2.4 GHz AMD Opteron 280
Memory	4 GiB
Storage	2 TB WD, 64 MB cache
Network	1 Gbit/s Ethernet
Operating system	Scientific Linux 4.6
JVM	Sun Java 1.6.0_17
Hadoop	Hadoop 1.0.0
Pig	Pig 0.10.0

Table 4: Characteristics of the 100 GB input subset.

Records	1 412 762 467
Unique trackers	38
Unique timestamps	3 985
Unique hashes	1 811 983
Unique swarms	2 290 161

time elapsed from the submission of the first query until the last-finishing query completes. The throughput of the processing system is defined as the ratio between the size of the data processed and the query execution time. The execution time of a query or job is the time elapsed from submission until completion. The resource utilization of the system is measured in terms of CPU, network, memory, and disk utilization. Disk utilization is measured as the number of I/O operations per second (IOPS), and the amount of data transferred to and from disk (MB/s), using the `iostat` tool.

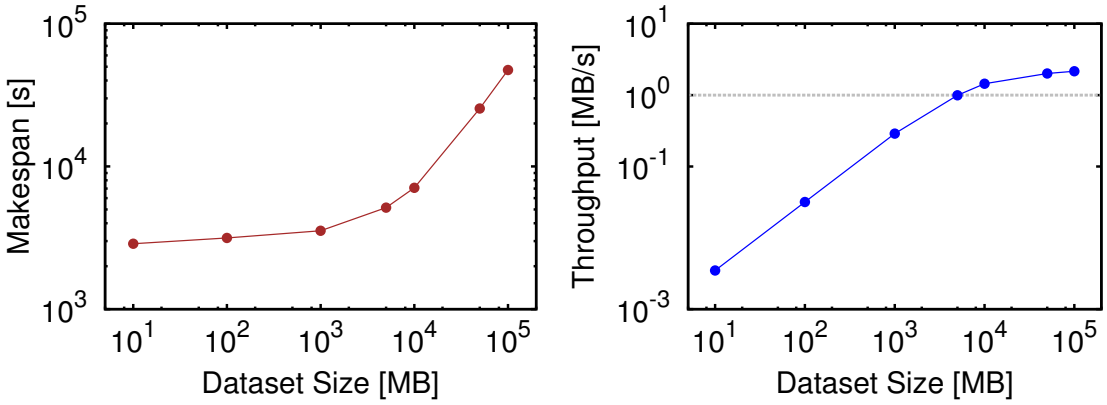
In our experiments, we determine the makespan and the throughput for increasingly larger subsets of the complete BTWorld data set, with sizes spanning the range 10 MB to 100 GB. These larger subsets include months of data for several BitTorrent trackers, and are thus representative for the types of measurement studies already published about BitTorrent [21, 23]. Table 4 summarizes the characteristics of the 100 GB BTWorld subset used in this work.

4.2 System-Level Performance Analysis

We evaluate the workflow makespan and system throughput for each subset size, with the results summarized in Figure 4. We find that the processing system delivers a stable performance of about 2 MB/s for the larger subsets tested in this work (5 GB and larger), with the corresponding increase in workflow makespan as the subset size increases. For the small data sets (10 MB to 1 GB), the limited amount of data also limits the available parallelism: there are only a few mappers and, up to 1 GB, there is only one reducer. The throughput, or the processing speed, increases steadily as the input data size increases, but is limited in our system to about 2 MB/s.

We turn our attention to the analysis of resource utilization, which is based on the processing of the 10 GB subset—the size after which the makespan increases almost linearly with the increase of the input, while the throughput remains stable. Figure 5 depicts the disk utilization of the cluster. We observe for both reads and writes that the peak bandwidth is much larger than the one-minute average bandwidth. This suggests that most data is transferred in short bursts. We also observe that our workload is relatively write-intensive around the one hour mark, which coincides with the `AH` and top-K queries.

As they are not the main focus of our big data study, we only discuss but do not depict the CPU, memory, and network utilization. The CPU and memory utilization are fairly constant



(a) Workflow makespan (b) Workflow throughput

Figure 4: The makespan and throughput for all input data sizes. The axes are logarithmic and do not start at 1.

during the execution of individual jobs, but vary greatly across jobs. The utilization of CPU and memory appear to be positively correlated, with overlapping peaks and valleys. The network utilization shows less consistent behavior, but is overall low; it reaches 100 Mbit/s for only a few seconds during the experiment.

4.3 Pig-Level Query Analysis

We analyze performance of the workflow described in Figure 3, first by comparing the performance of all queries, then by conducting an in-depth analysis of the queries selected in Section 3.3. We use for this analysis the 100 GB data set, which is the largest in our experiments and thus most stressful for the system under test.

We first compare the execution time of each query in this use case. Figure 6 depicts the query execution times which range from less than a minute (`SwT`) to several hours (`TKHL`). These variations stem from the size of the input data sets used by the queries as well as the complexity of their operations, as discussed in Section 3.3. We conclude that the BTWorld use case includes diverse queries, whose runtime can differ by several orders of magnitude.

We further study the impact on execution time of the workload parameter K, which impacts the Top-K type of queries (see Appendix A for more detail). We ran the `TKH` queries (local and global combined) with values of K ranging from 10 to 100 000, but the execution times (not depicted here) increased by just 12% while increasing K by four orders of magnitude. For `TKHL`, the majority of the time is spent on the first MapReduce job, which performs a grouping of the full data set and is unaffected by the value of K. We conclude that, for our implementation, the chosen value of K has little impact on the execution time.

We further investigate the Pig queries selected in Section 3.3. We compare their overall characteristics in Table 5, then discuss them in turn. Overall, the execution times of these queries are considerably different; however, an investigation at the level of the MapReduce jobs that implement each queries would obscure this difference. For example, the `AH` query is implemented automatically by the Pig system as a sequence of 2 MapReduce jobs, whose runtimes are of about 2 500 s and 5 500 s. A 2 500 s MapReduce job can also be observed as part of the set of jobs implementing the `TKTG` query. We conclude that an analysis at the level of Pig queries can reveal interesting information, which could be lost in a MapReduce-level analysis.

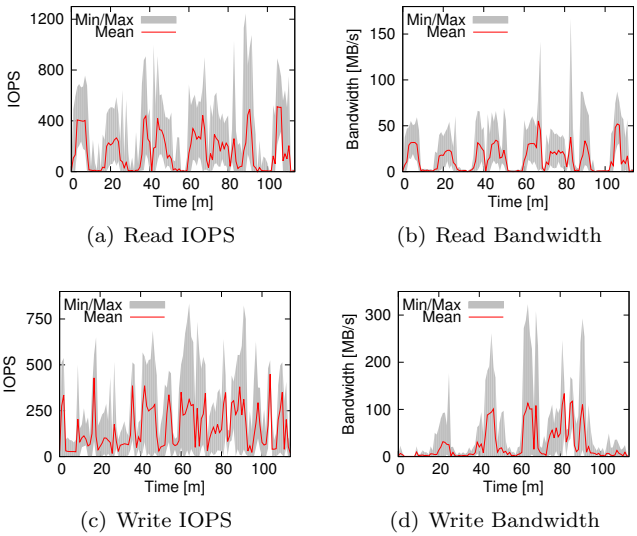


Figure 5: I/O utilization of the cluster sampled every second and aggregated per minute during the execution of the BTWorld workflow on the 10 GB data set. The gray areas represent the range of observed values.

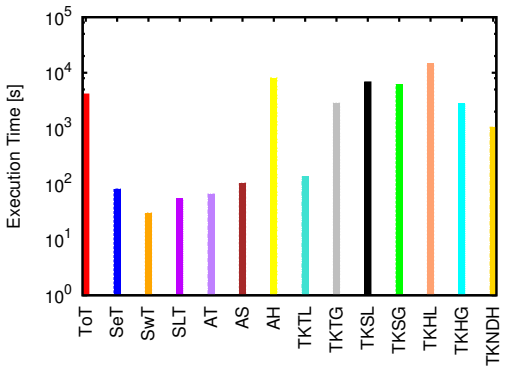


Figure 6: The query execution times for the 100 GB data set. Logarithmic vertical axis.

ToT. The `ToT` query is executed as a single map-heavy job. Only 6.66 MB of data are selected from the 100 GB input in the map phase (selectivity of about 1:6 000). The high selectivity can be attributed to the data set layout on HDFS. The input data on HDFS is grouped by `TR` and sorted by `TS`. As a result, grouping by `(TR, TS)` means that for most keys all input records are processed in the same map task. The map-side combiners can thus aggregate many records into one, before they are shuffled and sent to the reducers.

AS/AT. The `AS` and `AT` queries are some of the small post-processing queries used to extract useful information for statistical analysis from the generic data sets produced by queries such as `ToT`. They present an additional challenge in optimization as the number of maps and reducers cannot be tuned without negative impact on performance. For example, using more than one reducer only increases the overhead of launching tasks and spreading data.

AH. As the pseudocode in Listing 3 suggests, the `AH` query should ideally group the data

Table 5: Query characteristics. Starred queries (*) have inter-MapReduce-job dependencies.

Query Name	Query Execution Time (QET) [s]	MapReduce Jobs in Query	Execution Time Per Job [% of QET]
ToT	4 146	1	100
AS	104	1	100
AH*	8 110	2	70, 30
TKTG*	2 841	4	4, 4, 4, 88

set by timestamp and count the distinct hashes at every moment in time. However, a straightforward implementation in Pig, using the `DISTINCT` operator, fails at runtime. The `DISTINCT` operator within a grouping is executed by Pig in-memory in the reducers. For large groups this results in an out-of-memory exception and ultimately failure to complete the query. As a workaround, we have manually selected the distinct hashes by selecting distinct `(H, TS)` keys, grouping the result by `(TS)`, and counting the number of hashes per group. The resulting Pig query is split in two MapReduce jobs. The first job groups the data set by `(H, TS)` pair and outputs all distinct pairs. The second job groups the output of the first by timestamp, and utilizes combiners and reducers to count the number of hashes per timestamp. Most of the time (90%) is spent for this job in the mappers and combiners, similar to the `ToT` job.

TKTG. The global `TKT` query translates to a variety of MapReduce jobs. The query begins with three short jobs that create a list of the global top-K items, and finishes with a large map-only job performing a join of this list and the full data set. The latter job consumes most of the execution time for this query. Due to the replicated join support of Pig, the full list of top-K trackers is loaded into each of the mappers, and the join is performed map-side. For our chosen value of K the 100 GB input was reduced to 240 MB.

The performance of this query could be improved by choosing a MapReduce stack with support for indexes. In the final job of the global `TKT` query, the full data set is read to extract all data for the top-K trackers. With an index on the tracker field, only the data for the top-K trackers would have to be read from disk. The tracker-based directory structure of our data on HDFS provides an opportunity to read only specific directories as input to retrieve the data of specific trackers.

4.4 MapReduce-level Job Analysis

We analyze in this section the MapReduce jobs generated by the Pig system from our BTWorld workflow implementation; the results based on the run using the 100 GB data set are summarized in Figure 7. Overall, we find that the BTWorld use case results in MapReduce with diverse characteristics, in line with previous characterizations of MapReduce workloads.

Similarly to the execution times observed for Pig queries (Section 4.3), the job execution times (Figure 7(a)) span several orders of magnitude. However, half of the jobs take less than 2 minutes, which is consistent with the findings of Chen et al. [12, 13].

The MapReduce jobs exhibit various intra-job dependencies. Considering the duration of each phase of the computation depicted in Figure 7(b), we distinguish the following types of dependencies and summarize their presence in the entire MapReduce workload in Table 6:

- *Map-only*: performing join operations between two partitions of the data set.
- *Map-mostly*: performing aggregations in the map-phase with combiners.

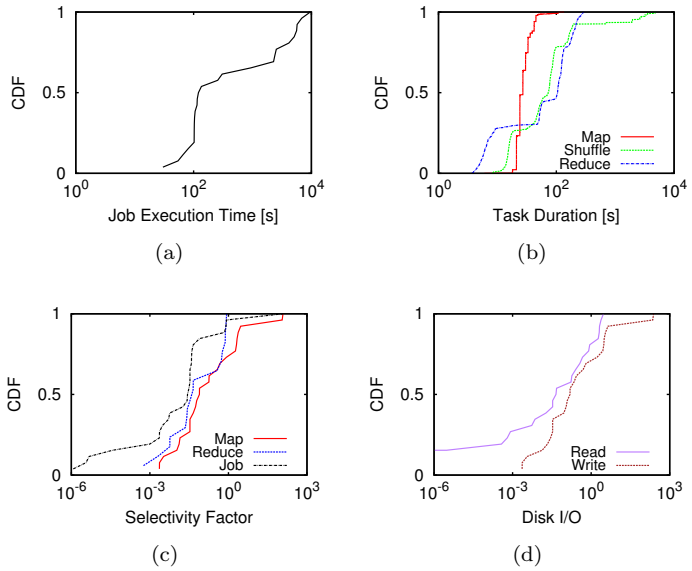


Figure 7: The MapReduce job profiles: a) the job durations distribution, b) the task durations distribution, c) the task selectivity (ratio between output and input size), and d) the disk I/O (bytes read and written) between tasks, normalized by input size.

Table 6: Types of MapReduce jobs, their presence in the MapReduce workflow, and SQL operator correspondence.

Job Type	Jobs in the MR Workflow	SQL Operator Correspondence
Map-only	5	Join, projection
Map-mostly	6	Map-side aggregation
Map-reduce	6	Filtering
Reduce-mostly	9	Reduce-side aggregation, projection

- *Map-reduce*: filtering the input data set, extracting and emitting the useful information to the next job.
- *Reduce-mostly*: performing aggregations in the reduce-phase (no map-side combiners).

We observe diverse I/O profiles for the MapReduce jobs in the BTWorld workflow. The reduce phase is statistically more selective then the map phase, rarely outputting more data than it receives as input (Figure 7(c)). Approximately 75% of jobs generate less intermediate data than the full size of the input data (Figure 7(d)). At the same time, there are a number of jobs that produce significantly more intermediary data, up to 100 times the size of the input data, which causes time-consuming I/O operations on the local file system.

5 Discussion

In this section, we discuss the usefulness of the BTWorld use case. We describe some of the lessons learned during the course of this study. We discuss the performance results of our empirical evaluation of BTWorld implemented using a MapReduce stack. Finally, we introduce the idea of extending the BTWorld use case towards a benchmark suite.

The main lesson we have learned from implementing the BTWorld use case is that the process extends beyond a trivial conversion of the P2P analyst's questions into queries that can be executed via a MapReduce stack. The data set layout needed adjustments to fit the storage layer. When designing the queries, performance improved by orders of magnitude by re-designing the Pig queries to re-use intermediary results across queries; however, this results in a more complex workflow and the need to make intermediary data persistent. The design of SQL-like queries in Pig was hampered by the inability of Pig to run `DISTINCT` operators without failing (see Section 4.3). Tuning a MapReduce system for a particular type of job, which is a common approach when the jobs perform very similar tasks, may be difficult for our use case—we have shown in Section 4.4 that our MapReduce jobs cover four broad, distinct categories. Although some adjustments we have made are specific to the chosen MapReduce stack (e.g. the `DISTINCT` operator failing in Pig), we have also gained insight that is applicable for a variety of processing platforms: that storing and re-using intermediary results may improve performance on many different platforms, that complex big data workflows may be challenging for today's data processing stacks, etc.

With a throughput of just 2 MB/s (see Section 4.2), the achieved performance seems poor. This can be partially attributed to the small cluster size, relative to the data set size, and also to the aged infrastructure of our testbed. We expect to obtain better performance by upgrading the system to a larger cluster, with more powerful nodes. However, several system-independent factors also contribute to the reduced performance: the input data is processed in 7 different jobs, multiple MapReduce jobs output gigabytes of data each, and the chosen MapReduce stack does not include indexing (in Section 4.3 we discuss several queries for which indexes would greatly reduce the amount of data read and thus the achieved performance). Modeling the performance of our use-case is non-trivial, because the runtime of each MapReduce job depends on input data and executed algorithm, and the MapReduce jobs are diverse in selectivity, structure, and data dependencies. We have also learned that understanding the way groups of MapReduce jobs operate as single high-level (Pig) queries is also important in engineering performance.

To create an industry-wide benchmarking suite, we have started and we currently lead within the SPEC organization[1] a joint, community-wide effort on benchmarking time-based analytics

[1]The Standard Performance Evaluation Corporation (SPEC) includes the SPEC Research Group (RG), which aims to "foster the interaction between industry and academia in the field". We conduct work in the Cloud Work-

Table 7: The BTWorld use case vs. state-of-the-art MapReduce benchmarks and use cases.

	Queries/Jobs	Diversity	Data Set	Layout	Volume
MRBench [15]	business queries	high	TPC-H	relational	3 GB
N-body Shop [14]	filter/correlate data	reduced	N-body sim.	relational	50 TB
DisCo [6]	co-clustering	reduced	Netflix [28]	matrix	100 GB
MadLINQ [7]	matrix algorithms	reduced	Netflix [28]	matrix	2 GB
ClueWeb09 [29]	web search	reduced	Wikipedia	html	25 TB
GridMix [16] PigMix [17]	artificial	reduced	random	binary text	variable
HiBench [30] PUMA [31]	text/web analysis	high	Wikipedia	binary text html	variable
WL Suites [12]	production traces	high	-	-	-
BTWorld	**P2P analysis**	**high**	**BitTorrent**	**relational**	**14 TB**

platforms. The use case presented in this work is a basic building block for such a benchmark, including data structure and content, and a full time-based analytics workflow. However, numerous challenges still need to be addressed: defining both full-data-set and incremental processing components; defining multiple workflows and possibly entire workloads; creating realistic input generators; devising metrics for various non-functional system properties (e.g., reliability, elasticity, energy-friendliness); etc.

6 Related Work

We have already discussed two studies of MapReduce workloads from global-scale companies [12, 13]. We survey in this section six MapReduce use cases and five representative benchmarking suites. In contrast with these approaches, which are summarized in Table 7, our BTWorld use case focuses on a new application domain (system management, computer science), and combines exhibiting high workload diversity (various types of dependencies and operational profiles) with a large-volume data set.

Closest to our work, **MRBench** [15] implements in MapReduce the queries included in the TPC-H [32] database benchmark, which are representative for decision support. MRBench executes complex business oriented queries with concurrent data modifications on 3 GB of relational data. BTWorld is considerably different in scope, application domain, data, and workload. Also close to our work, the **N-body Shop** group [14] analyzes massive amounts of data representing the astrophysics application domain. The workload filters and correlates data at different moments of time, using selection, join, and projection operators, which target roughly 36 GB. BTWorld exceeds the scope of N-Body Shop with a broader range of algorithms and a larger and more complex workflow.

Much of the remaining previous work focuses on matrix algorithms [6] [7], web search [29], saturation tools with rather unrealistic workloads [16] [17], scalable data generation from real workload traces [12], and various individual MapReduce applications [30] [31].

ing Group, which is an SPEC RG branch that aims to develop the methodological aspects of cloud benchmarking, including services for big data processing.

7 Conclusion

Various scientific and industrial processes rely on being able to automatically process large amounts of periodically collected data, but currently only few use cases describe such workloads. In this work, we have introduced from a big data perspective BTWorld, a big data use case representative for time-based analytics.

BTWorld aims at collecting, processing, and preserving large amounts of periodic measurement data representing the operation of the global BitTorrent network, which accounts for a third of the global upstream traffic in 2013; thus, BTWorld enables novel big-data-driven research in computer science. We have described the use case, focusing on the BTWorld data set and on several research questions that BTWorld can answer. We have designed for the MapReduce stack a logical workflow, which includes a data layout and SQL-like queries that can answer the research questions efficiently. The BTWorld workflow includes diverse queries and jobs, which represent well three types of data dependency—inter-query, inter-job, and intra-job. We have conducted an empirical study of the logical workflow implemented in the Pig–Hadoop–HDFS MapReduce stack; and analyzed system-level performance, and the performance of Pig Latin queries and their corresponding MapReduce jobs. We have shown evidence that the BTWorld workflow qualitatively and quantitatively extends the state-of-the-art in understanding MapReduce workloads; for example, the MapReduce experiments exhibit challenging features: three or more orders of magnitude differences in data data sizes per observed item, data selectivity, and job duration. Last, we have discussed the usefulness of the BTWorld use case, including a path towards a benchmarking suite, through the help of our SPEC Cloud WG partners.

For the future, we aim to include in the use case the various semi-structured and unstructured data we have also collected for BTWorld, such as comments of users on shared content.

Acknowledgment

The first author works through the Honours Programme (Challent) of the Delft University of Technology. This publication was supported by the Dutch national program COMMIT, STW/NWO Veni grant 11881, and EU FP7 projects P2P-Next (grant 216217) and QLectives (grant 231200).

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," *Comm. of the ACM*, vol. 51, no. 1, pp. 107–113, 2008. 4, 8

[2] T. White, *Hadoop: The Definitive Guide*. Yahoo Press, 2010. 4, 8

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *MSST*, pp. 1–10, IEEE, 2010. 4, 8

[4] K. Yelick, S. Coghlan, B. Draney, R. S. Canon, *et al.*, "The Magellan report on cloud computing for science," *US DoE*, 2011. 4

[5] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *OSDI*, pp. 29–42, USENIX, 2008. 4

[6] S. Papadimitriou and J. Sun, "Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining," in *ICDM*, pp. 512–521, IEEE, 2008. 4, 18

[7] Z. Qian, X. Chen, N. Kang, M. Chen, Y. Yu, T. Moscibroda, and Z. Zhang, "MadLINQ: large-scale distributed matrix computation for the cloud," in *EuroSys*, pp. 197–210, ACM, 2012. 4, 5, 18

[8] M. Wojciechowski, M. Capotă, J. Pouwelse, and A. Iosup, "BTWorld: Towards Observing the Global BitTorrent File-Sharing Network," in *LSAP Workshop in conjunction with HPDC*, ACM, 2010. 4, 5, 7

[9] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, pp. 103–111, Aug. 1990. 4

[10] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool," *Commun. ACM*, vol. 53, no. 1, pp. 72–77, 2010. 5

[11] Apache Hadoop Wiki, "Powered By Hadoop." 5

[12] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The Case for Evaluating MapReduce Performance Using Workload Suites," in *MASCOTS*, pp. 390–399, IEEE, 2011. 5, 15, 18

[13] Y. Chen, S. Alspaugh, and R. Katz, "Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads," *Proc. of the VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, 2012. 5, 15, 18

[14] S. Loebman, D. Nunley, Y.-C. Kwon, B. Howe, M. Balazinska, and J. P. Gardner, "Analyzing massive astrophysical datasets: Can Pig/Hadoop or a relational DBMS help?," in *Cluster*, pp. 1–10, IEEE, 2009. 5, 18

[15] K. Kim, K. Jeon, H. Han, S.-g. Kim, H. Jung, and H. Y. Yeom, "MRBench: A Benchmark for MapReduce Framework," in *ICPADS*, 2008. 5, 18

[16] "The GridMix Hadoop Benchmark." 5, 18

[17] "The PigMix benchmark." [Online] `http://wiki.apache.org/pig/PigMix`. 5, 18

[18] Sandvine, "Global Internet Phenomena Report 1H2013." 5

[19] D. Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," in *SIGCOMM*, ACM, 2004. 6

[20] B. Cohen, "The BitTorrent Protocol Specification." 6

[21] 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, 12

[22] M. Meulpolder, L. D'Acunto, M. Capotă, M. Wojciechowski, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips, "Public and private BitTorrent communities: A measurement study," in *IPTPS*, USENIX, 2010. 7

[23] B. Zhang, A. Iosup, J. A. Pouwelse, and D. H. J. Epema, "Identifying, analyzing, and modeling flashcrowds in BitTorrent," in *P2P*, IEEE, 2011. 7, 12

[24] R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding availability," in *IPTPS*, pp. 256–267, Springer, 2003. 7

[25] T. Karagiannis, A. Broido, N. Brownlee, kc claffy, and M. Faloutsos, "Is P2P dying or just hiding?," in *GLOBECOM*, pp. 1532–1538, IEEE, 2004. 7

[26] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," in *SIGMOD*, pp. 1099–1110, ACM, 2008. 8, 11

[27] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a High-Level Dataflow System on top of Map-Reduce: the Pig Experience," *Proc. of the VLDB Endowment*, vol. 2, no. 2, pp. 1414–1425, 2009. 8, 11

[28] "Netflix prize." 18

[29] "The ClueWeb09 Dataset." 18

[30] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The Hibench Benchmark Suite: Characterization of the MapReduce-based Data Analysis," in *ICDEW*, pp. 41–51, IEEE, 2010. 18

[31] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "PUMA: Purdue MapReduce Benchmarks Suite," tech. rep. 18

[32] TPPC, "TPC-H Benchmark Specification," 2008. www.tpc.org/tpch. 18

Table 8: Improvements in the BTWorld workflow, impacting performance and/or robustness.

Keywords	Solved?	Impact
Duplicate operations, Reusing intermediary data	Solved	Performance
Memory overcommitment, DISTINCT operator	Solved	Robustness
Redundant reads, Top-K selection	Unsolved	Performance
Tweaking configuration, Workflow diversity	Unsolved	Both

A Improvements

Throughout the design and implementation of the BTWorld workflow for MapReduce, we have made several improvements to increase the performance and robustness of our workflow. We present in Table 8 the most relevant issues we have identified and their current status.

Reusing intermediary data. Our first workflow design was based on a one-to-one conversion of analyst questions to queries, without any inter-dependency between them. Thus, many computations where repeated in different queries. We identified the duplicate operations and stored their ouput once, such that subsequent queries can reuse it.

During the instantiation of the BTWorld workflow we have identified several cases in which reusing intermediary data led to performance gains. For example, the `TKNDH` query was originally split into two subqueries which calculate the number of newborn hashes (i.e., the first appearance of a hash) and the number of dead hashes, respectively. We found empirically that the most time-consuming operation in both queries was grouping the input by hash. By combining the two queries into one, the grouping operation is performed only once, and the generated output is used twice to extract the newborn and dead hash counts. This particular improvement led to a decrease in query execution time of 48%, from 1 988 seconds for two separate queries to 1 051 seconds for the combined query.

Crash of nested DISTINCT. While most improvements were made to increase the performance of our workflow, some were necessary in order to guarantee completion of our experiments. As we scaled up our experiments from 10 MB up to 100 GB, the `AH` query using the `DISTINCT` operator crashed. The particular query involves grouping data by timestamp and finding the distinct hashes within each group. Our implementation of this query used the `GROUP BY` and `FOREACH` operations to group our data, and we used the `DISTINCT` operator inside the `FOREACH` block to find the unique hashes. Pig executes this construct by grouping in the map and shuffle phases, and performing the `DISTINCT` operation in-memory in the reduce phase. For this particular operator, overcommitting memory in reducers led to catastrophic failure.

Through experiments we determined the breaking point between 50 GB and 100 GB of input. However, the crash is not directly related to the input size, as the crash occurs when a single group in the input is too large. Further investigation of the exact breaking point for this operator is planned as future work. For our experiments we have chosen an alternative implementation without the `DISTINCT` operator (see section 4.3).

Impact of K on Top-K query performance. As part of our experiments, we have evaluated the impact of the value K on the performance of Top-K queries. In particular, we have measured runtimes for the `TKH` queries (local and global combined) with the value of K ranging from 10 to 100 000. The observed runtimes are depicted in table 9, including the runtimes for the two largest jobs in the queries. The first `TKH-L` job consists of reading the full data set, grouping it by `(H, TS)`, and a final aggregation. This operation is independent of the value of K, which supports our observed constant runtime for this job. Although performance could be improved

Table 9: Runtimes with varying K for all TKH jobs combined, the first TKH-L job, and the second TKH-L job.

K	Runtime [s]	Runtime TKH-L-1 [s]	Runtime TKH-L-2 [s]
10	17 073	9 581	4 985
100	17 079	9 536	5 040
1 000	17 138	9 509	5 055
10 000	17 333	9 490	5 052
100 000	19 093	9 529	5 588

significantly if the amount of redundant data read is reduced (e.g., by utilizing indexes), we have not addressed this issue in our implementation.

Tweaking MapReduce configuration. For many MapReduce applications, tweaking the configuration of the MR stack can improve performance. Predictable characteristics in a workload (e.g., a model of the input) can be used to help determine the (near-)optimal configuration for a specific application. In the BTWorld use case we have observed a high diversity in the input, which can have significant impact on performance (e.g., by affecting the size of groups in a GROUP BY operation). Furthermore, the workflow is both diverse and complex, with several distinct job types and varying job lengths. Due to this diversity, optimizing for one query may be counter-productive for the performance of different types of jobs. We do not have a model, but empirically we have found that the default configuration works well for our workload.