Keddah: Capturing Hadoop Network Behaviour

Jie Deng, Gareth Tyson, Felix Cuadrado and Steve Uhlig
School of Electronic Engineering and Computer Science
Queen Mary University of London
Email: {j.deng,gareth.tyson,felix.cuadrado,steve.uhlig}@qmul.ac.uk

***Abstract*—As a distributed system, Hadoop heavily relies on the network to complete data processing jobs. While Hadoop traffic is perceived to be critical for job execution performance, the actual behaviour of Hadoop network traffic is still poorly understood. This lack of understanding greatly complicates research relying on Hadoop workloads. In this paper, we explore Hadoop traffic through experimentation. We analyse the generated traffic of multiple types of MapReduce jobs, with varying input sizes, and cluster configuration parameters. As a result, we present Keddah, a toolchain for capturing, modelling and reproducing Hadoop traffic, for use with network simulators. Keddah can be used to create empirical Hadoop traffic models, enabling reproducible Hadoop research in more realistic scenarios.**

I. Introduction

MapReduce [1] automatically spreads computation across multiple nodes, substantially lowering the adoption barrier for parallel computing on very large datasets. Abstracting the complexity of parallel computation from users has greatly fostered multiple research in this area. A significant amount of work explores novel data intensive programming languages and computation models, as well as efficient schedulers [2], or efficient parallel implementations of machine learning algorithms. However, the *network* component of MapReduce has been neglected, despite its integral role in distributed computations.

Many possible innovations are possible with datacentre networks supporting Hadoop clusters. Yet, work has been limited to a few topics, e.g., router queues [3] or SDN controllers [4]. A key reason is that, currently, it is difficult to experiment with novel network technologies with a Hadoop cluster. For instance, the network interactions resulting from Hadoop are dynamically affected by the algorithm and dataset, making it impossible to profile from the codebase. Building a dedicated cluster for experimentation is prohibitively expensive, while gaining meaningful workload data can be extremely challenging. Although recent attempts to share workload data from clusters have been established (e.g., SWIM [5]), only high level statistic counts are available.

The above factors make it difficult to evaluate new network technologies (real or simulated) in Hadoop clusters (e.g., topologies, routing protocols, transport protocols). This is further exacerbated by a generally poor understanding of how Hadoop actually behaves at the network layer (§II). In this paper, we focus on two key research questions: (i) What are the network traffic characteristics of Hadoop jobs? and (ii) How can we recreate realistic Hadoop traffic patterns for simulated experiments? In this paper, we answer these questions by characterising Hadoop traffic (§IV), so to formulate empirical models that can be used to reconstruct traffic in simulators. We built a tool, Keddah (§V), that makes it possible for cluster operators to capture and automatically create these empirical models for use by the research community, thereby allowing simulated experiments with new layer 2/3/4 technologies. Keddah is freely available to the research community, alongside our pre-generated traffic models.[1]

II. Background and related work

A. Hadoop MapReduce

Hadoop MapReduce is a system designed to support parallel data processing across multiple compute servers. It consists of a distributed storage system, the Hadoop Distributed File System (HDFS), and a distributed execution framework called YARN [6]. Hadoop involves a two stage execution flow where a 1-to-1 *mapping process* is followed by a many-to-1 *reduction process*. Jobs in Hadoop start with the YARN Resource Manager allocating *map* and *reduce* roles to different nodes, instructing them to perform a given computation on a given set of data blocks and intermediate keys, respectively. Following this, mappers read the appropriate data from HDFS. Once each mapper has retrieved its allocated data blocks, it begins to execute the computation (e.g., sorting). All the results from mappers are then transferred during the *shuffle* step to the reducer nodes partitioned by key. Each reducer node aggregates the individual results, and once all computation has been completed, the reducer writes the result back to HDFS.

B. Related work

Just a small number of studies have investigated the network behaviour of mainstream "Big Data" platforms

[1] https://github.com/deng113jie/keddah

such as Hadoop. For instance, [7] discusses the performance of Spark, finding that job completion time can be reduced by only 2% through network improvements. A more recent article [8] pointed out that these results are partially an artifact of specific platform optimisations of Spark (namely, heavy data replication to avoid network traffic); in contrast, they found that PageRank jobs can be improved up to 3x by increasing the network capacity from 1 Gbps to 10 Gbps. Other works have tried to improve Hadoop performance by, for example, performing aggregation on network devices in the cluster [9]. These works have confirmed the vital role of the network in the performance of distributed processing platforms such as Hadoop MapReduce.

Various studies have looked at Hadoop workloads, primarily for the purposes of cluster benchmarking (e.g., comparing job completion time across clusters). Hadoop provides a number of benchmarking tools. TeraSort [10], later standardised as TPCx-HS [11], is the most popular benchmarking job for cluster comparison. There are also more diverse Hadoop benchmarking projects such as the Big Data benchmark [12] and HiBench [13]. These provide a more diverse set of Big Data processing use-cases, such as machine learning or graph processing (e.g., HiBench uses Pegasus [14] to benchmark PageRank). There are however no studies that uses these benchmarks to investigate the behaviour and role of the network in Hadoop jobs. Previous simulation approaches profiled the resource usage in Hadoop, including task prediction [15], computation resources (CPU, memory, storage [16]) and computation time prediction [17], [18]. To the best of our knowledge, we are the first to profile and model Hadoop network traffic.

III. Testbed and Methodology

To extract the network patterns of a Hadoop cluster, it is necessary to measure its live behavior. For this, we use well known benchmarking tools to execute a number of different jobs on a real Hadoop cluster. Packet traces are then collected across the cluster to study, characterise and model the traffic. Our methodology allows us to sample the network traffic over a set of variables, including cluster settings (number of replications, HDFS block size, cluster size), type of jobs and job settings (output replication, number of reducers and job specific parameters). We sample a diversity of Hadoop jobs, including machine learning, graph algorithms, and scientific computation. We focus on three types of jobs [9], [19], taken from the benchmarking tool HiBench [13]. The jobs are (*i*) *TeraSort*: the TPCx benchmarking job for Hadoop clusters [11]; (*ii*) *PageRank*: a graph processing operation for calculating the weight between vertices [14]; and (*iii*) *kmeans*: one of the most popular data clustering algorithms [20]. We considered other types of jobs in our initial experiments (*e.g.,* word count, join queries, bayes), and found the selected three provide a good sample of different types of MapReduce computation.

To collect the traffic, we ran the jobs in two clusters: a 16 nodes physical cluster connected to the same switch, with each node containing 8 cores and 32GB memory; and a virtual cluster from Amazon Web Services (AWS) with up to 30 nodes and hardware resources. Collectively, these clusters can run jobs requiring up to 500GB of memory and 6TB of storage. Given that the datasets used in our experiments are less than 20GB, these clusters are large enough for our purposes. Our Hadoop codebase is built upon Cloudera Distributed Hadoop version 5.4.7. We rely on the default settings for Hadoop and the jobs (unless stated). An sflow agent is installed on each machine with a dedicated sflow collector on the same switch to obtain the traffic statistics.

To obtain an accurate coverage of the network behaviour, it is necessary to run jobs multiple times. For example, data placement is not deterministic in Hadoop,[2] and therefore multiple runs are necessary to sample different data placements. To get an idea of the number of runs required, we launch 1000 identical TeraSort jobs. Each time we clean the cache and calculate the aggregated traffic volume generated. We then use the Kolmogorov-Smirnov (KS) test to check if the traffic volume distribution after a certain number of runs is similar to the one found with 1000 samples. Figure 1 presents the p-value and distance of the KS test after each run. There is significant variance when performing fewer than 200 runs. However, we find that the p-value is close to 1 (with a very small distance) after 400 runs, indicating that a similar distribution is found. The same convergence in distribution happens for all traffic attributes in the paper, such as number of sessions, and number of nodes. Therefore, all our subsequent traffic analysis is based on 400 runs for each individual setup.

IV. Understanding Hadoop traffic

We begin by exploring the traffic generated by several Hadoop jobs. Our overall aim is to gain a general understanding of Hadoop traffic, to enable us later to (re-)generate similar traffic.

A. Traffic Decomposition

Hadoop generates several types of traffic, including HDFS control messages, HDFS data transmissions, Shuffle data transfers, and Yarn control messages (e.g., keep alives). We observed that from both virtual and physical clusters, the amount of control plane traffic is negligible, with in excess of 99% of traffic coming from HDFS data transfers and Shuffle. Hence, we ignore

[2]See chooseRandom in org.apache.hadoop.net.NetworkTopology.

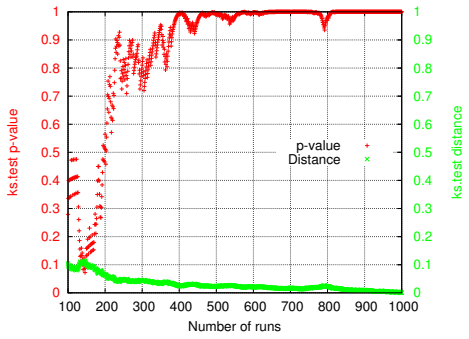


Fig. 1: Convergence (measured as p-value and distance) of distributional properties, through KS test against a distribution based on 1000 samples.

TABLE I: Traffic composition of Hadoop jobs.

Job	HDFS import	HDFS export	Shuffle
sort	0.01	0.7	0.29
wordcount	0.03	0.66	0.31
TeraSort	0.1	0	0.9
Bayes	0.54	0.46	0
kmeans(itr-1)	0.31	0.67	0
kmeans(itr-2)	0.07	0.82	0
PageRank(itr-1)	0.10	0.66	0.23
PageRank(itr-2)	0.25	0.09	0.66

control traffic. Table I presents the average fraction of different traffic components across all Hadoop jobs considered when using the default settings on a 16 nodes cluster. The diversity across jobs is significant. For example, we observe no Shuffle traffic in map-only jobs such as Bayes and kmeans compared to 90% in TeraSort.

It is also possible to inspect how each of these traffic components is generated across stages of a job. By definition, each stage occurs sequentially: *HDFS Read* loads the data onto mapper nodes, *Shuffle* exchanges data between the mapper/reducer nodes. Finally, *HDFS Write* stores the result. Due to space constraints, we focus on how these traffic components are generated in three example jobs (kmeans, TerraSort and PageRank). Figure 2 presents the average amount of traffic generated per-second on the physical cluster for: the second iteration of PageRank jobs (with 10 million vertices), TeraSort jobs with 6GB data, and the first iteration of kmeans jobs with 14 clusters and 14 million samples per cluster. It can be seen that each job exhibits broadly similar trends, but with key differences.

Closer inspection of the trends reveal that these differences emerge from the varying ratio between the three core traffic components. For instance, TeraSort's first 10% of traffic is HDFS Read, followed by the remaining 90% that corresponds to Shuffle traffic. Similarly for PageRank, there is 25% HDFS Read at the beginning,

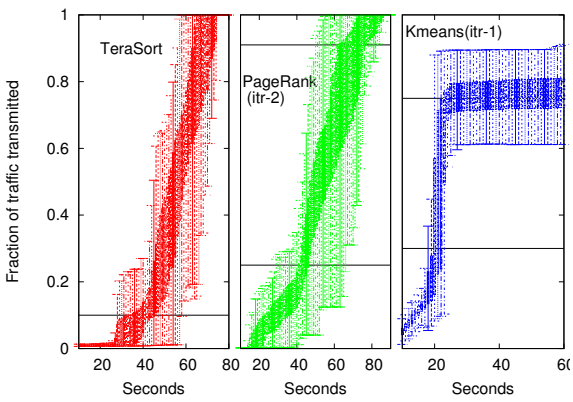


Fig. 2: Fraction of traffic transmitted over time, for TeraSort, PageRank and Kmeans. The horizontal lines demarcate HDFS Read, Shuffle and HDFS Write traffic.

then 66% Shuffle, and finally the last 9% is HDFS Write. All traffic seen on the network follows this three stage sequence. Clearly, this stability in the structure is helpful for modelling traffic. Still, the variability over different runs for a given type of job is quite high (due to the randomness in data placement). Therefore, it is necessary to run jobs at least 400 times (c.f., Figure 1) to capture the full range of behaviour.

B. Impact of Dataset Size

We next inspect the impact that the input dataset has on traffic volumes generated by each job.

1) Kmeans: We run kmeans with 10 clusters and datasets ranging from 500k to 20m samples (generated using HiBench). The assignment step and update step are treated as separate jobs in Hadoop. Figure 3(a) presents the amount of traffic generated per run, while varying the dataset size. The first to the penultimate, and the last rounds are shown. It can be seen that traffic increases as a linear function of the dataset size. We see this across all iterations with the exclusion of the last. This can be easily explained through inspection of the source code: All iterations but the last require assigning the samples to each cluster, while the last is only concerned with the final computation of the clusters, which generates a fixed volume of traffic (as we use 10 clusters throughout).

2) TeraSort: We run TerraSort over randomly generated data files ranging from 1GB to 20GB (generated using TeraGen). Again, Figure 3(b) shows that the traffic volume exhibits a linear trend with respect to the data file size. Surprisingly, though, the variability is quite high, even for identical job settings (about a quarter of the file size, similar to Figure 4). To better understand this, we experimented with both the computationally worst-case and best-case datasets, but found no difference. The reason lies in the fact that the data is processed

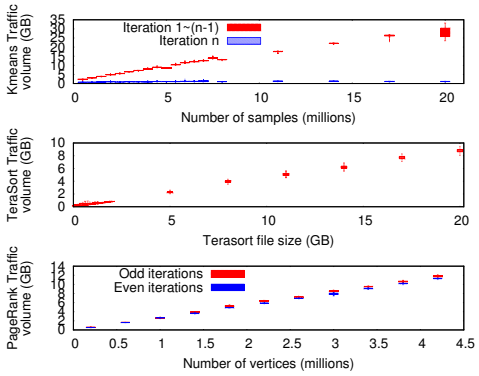


Fig. 3: Total traffic volume over various jobs/data inputs.

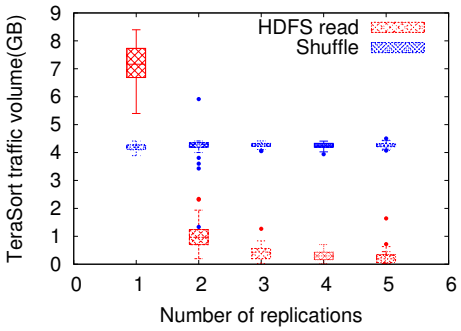


Fig. 4: Volume of HDFS Read and Shuffle traffic for TeraSort 4GB jobs over various replications. Increasing HDFS replications in the cluster does not necessary decrease HDFS traffic.

in a distributed manner, and is spread out across the cluster irrespective of whether it is already sorted or not. Therefore, for the traffic seen on the network, how well the data is sorted beforehand is irrelevant. The positive aspect, however, is that to model TeraSort traffic, only the data size needs to be considered (not how well this data is already sorted).

3) PageRank: Finally, we run PageRank over graph data containing between 2 million (1.1GB) and 10 million vertices (5.5GB), with the number of edges following a Zipf (α=0.5) distribution with average degree of 40. PageRank in Hadoop adopts two stages. First, it generates partial matrix-vector multiplication results; then, it merges the multiplication results. The sequence of both stages run iteratively until the rank converges. Despite this, again, Figure 3(c) confirms that traffic volume is a linear function of dataset size. We have also repeated this process across several other jobs, to consistently witness this linear relationship.

C. Cluster Settings

Next, we explore how the traffic changes when we modify the cluster settings. We have profiled parameters including replication factor, cluster size, block size, split size and number of reducers. We do not consider settings that usually only affect the computation performance rather than network (e.g., Java memory allocation [7]) at this stage, leaving their exploration as future work.

Replication factor is a well known factor that affects the Hadoop performance [21], [22], however the traffic is not well studied yet. We focus on native Hadoop methods instead of techniques proposed by research community [23], as built-in Hadoop concepts such as replication and block size are more fundamental and more important at this stage.

HDFS stores data on nodes across the cluster. These nodes may also be used as mappers; hence, a mapper that already contains the data locally generates no extra traffic. To improve resilience, HDFS has a replication setting that allows the same block to be stored on multiple nodes. Intuitively, the higher the replication level, the lower the traffic volume.

To explore the effect of replication, we run a new set of TeraSort (4GB) jobs, varying the replication levels. Figure 4 presents the aggregate traffic volume generated when varying the replication rate between 1 and 6. It can be seen that 86% of the traffic is removed when increasing the replication rate from 1 to 2. A replication level of 3 further decreases it by 56%; and by another 23% for 4 (additional replication suffers from diminishing returns). In our cluster of 15 nodes, with the maximum replication of 15, no HDFS Read traffic is seen. Note, however, that the savings come with a startup cost, as it is necessary to preload the data.

The next parameter we consider is the number of nodes in the cluster. This may impact network traffic significantly, as nodes will need to hold more blocks if fewer nodes are available. We run TeraSort over a 1GB dataset while varying the cluster size. Overall, the total traffic volume follows the same distribution across all cluster sizes, though the traffic generated decreases when moving from 6 to 10 nodes by 20%, and then remains roughly static (3% variance) afterwards. With more nodes holding the dataset in HDFS, local mapper assignment becomes easier. However, for a fixed input data size, the number of mappers as well as the number of map tasks is deterministic with the same input size. For instance, a 1GB TeraSort job will utilise a maximum of 8 nodes. This presents an advantageous property for traffic modeling: the traffic for TeraSort 1GB jobs on a 15 node cluster will be identical to that of any other cluster size above 8. We further confirmed this on other jobs such as PageRank and kmeans. This enables us to predict the generated traffic, as long as we know the cluster capacity is above a certain threshold.

Finally, we profiled three parameters that directly

affect MapReduce performance, including HDFS block size (affecting HDFS file distribution), mapper input split size (affecting the number of mappers) and the number of reducers. These parameters affect the network traffic in an expected way; the same amount of total traffic is generated, but it is split differently across the cluster nodes. Moreover, the impact of varying each parameter is localised to a specific traffic stage. For example, by increasing HDFS block size linearly, the number of blocks will reduce accordingly, but the shuffle volume is not affected at all. This way, we reduced the sampling space when capturing Hadoop traffic with various parameters, as each parameter can be profiled separately.

In this section, we profiled Hadoop traffic over a set of variables. Those variables selected are factors directly affect the Hadoop performance and commonly configured in practise, including physical cluster size and resource; Hadoop cluster configurations like number of replications, HDFS block size and number of reducers; and various Hadoop jobs with different job parameters. Also with the aim of reproduce the Hadoop traffic, other settings can be captured by the same process as we will show in the following section.

V. KEDDAH: HADOOP TRAFFIC GENERATION

The previous section has characterised Hadoop traffic. Here, we present Keddah, a tool to capture, model and therefore reproduce Hadoop traffic, so that it can be used by simulators.

A. Keddah methodology

Our goal is to create flow-level traffic models, to allow realistic Hadoop traffic to be replayed in a simulator. To achieve this, we have devised Keddah, a toolchain that can execute real Hadoop jobs in a cluster, capture the traffic and then generate appropriate empirically-derived models. These can then be shared and passed into a simulator to replay the traffic. Figure 5 presents the workflow. Keddah can be deployed in any cluster in which traffic traces can be collected.

Keddah accepts a list of jobs and dataset parameters (see §IV-B). It then launches Hadoop jobs in the cluster for each job and parameter combination, and repeats each combination a default of 400 times (see Figure 1). Upon each run, traffic traces are collected[3] and reported back to the controller. The following factors are then extracted from the network traffic traces of each job execution:

- *Number of source nodes generating flows*: This is dictated by the Yarn resource manager together with the MapReduce application master. Both elements automatically select the set of nodes (mappers and reducers) from the available cluster resources. Importantly, the number is *not* defined by the size of the cluster (which only sets an upper limit): A small job will run on a small number of nodes regardless of the cluster size. The values are non-deterministic, as random data placement means that data will be stored on different HDFS nodes upon every job execution. Indeed, mappers only request data from other nodes if the split is not available locally.
- *Number of destination nodes accepting flows*: This is the number of nodes that accept connections, and differs from the number establishing connections. For instance, theoretically, due to random data placement, a single node could contain all data chunks, while multiple other nodes request data from it.
- *Number of flows generated*: This is a product of the random data placement, dataset size and the nature of the job.
- *Aggregated traffic volume*: This is simply a product of the number of flows and their individual sizes.
- *Time distribution of flows*: This is defined by the time when the first packet of a flow is observed. We measure the start-time relative to when the job begins its execution. Although, in theory, flows should start at the beginning of the map or reduce tasks, we find that the start times vary quite a lot in practice. This is due to different node CPU speeds, and different times for mapping tasks.

Collectively, these five attributes can be used to characterise the application-layer network activity of Hadoop. After they have been collected across all 400 runs (of the same job/dataset), the empirical distributions of each of these five factors are computed. Importantly, as the three traffic components (HDFS Read, Write and Shuffle) exhibit very different behaviors (see §IV), we compute the five factors separately for each. An extra consideration is that some jobs run multiple iterations. In PageRank, for example, weights in the graph are adjusted repeatedly after aggregating results and re-generating the new graph. In Hadoop, this is performed by creating a sequence of multiple separate jobs (one for each iteration). This means that it is necessary to model iterative jobs as several separate sequential jobs with a certain interval between them. Keddah automatically detects iterative jobs and records the interval.

Once the five empirical distributions for each of the traffic components have been computed, Keddah uses standard model fitting techniques to convert them into probability models. By default, Keddah evaluates the empirical distributions against several models: normal, Poisson, exponential, gamma, chi-squared, Weibull and Pareto (more can be plugged in). In each case, the

[3]Keddah is agnostic to how traffic is collected; it can either be on the end hosts or the switches.

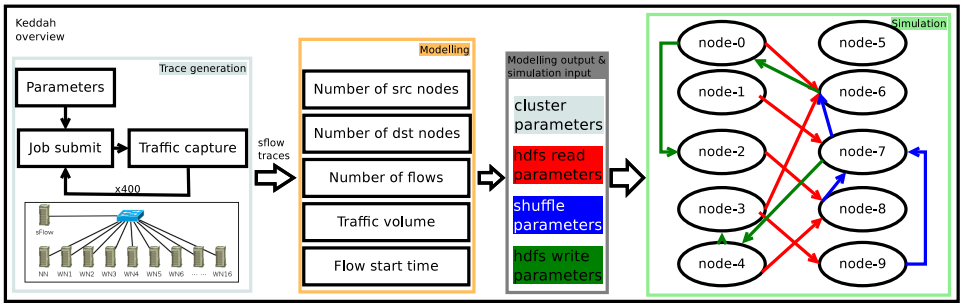


Fig. 5: keddah workflow, starting with execution of real jobs and ending in the generation of the traffic in a simulator.

empirical distribution is evaluated against all possible models to parameterise the best fit (based on highest p-value and lowest distance obtained from the KS test). The output is a file containing three separate groups of model parameters for HDFS Read, Write and Shuffle. Each group contains the probability distribution model (and parameters) for each of the 5 attributes. In essence, this is a compact way of recording application-layer traffic (rather than directly replaying traces). We term this a *Keddah model file*.

Keddah model files can then be loaded into ns3 via a plug-in we have developed. We emphasise that these models should not be generalised, as they are extracted from empirical models that only reflect the traffic they were computed from. Thus, each job/dataset will have its own individual file representing the traffic seen within the cluster that generated it. Our intention is to allow any Hadoop operator to contribute statistical traffic models, allowing researchers to simulate the corresponding traffic. Critically, by abstracting the traces via parameterised models, operators are protected from revealing sensitive information (e.g., network configuration).

B. Results

We have used Keddah to capture and model traffic across a wide variety of jobs and datasets in our testbeds. In each case, Keddah has launched the jobs, recorded the traffic and produced the traffic models. For brevity, we present the results from three example jobs, although the results are similar across all other runs we have experimented with. Figure 6 presents the empirical vs. modelled distributions across the four attributes (due to space constraints, we only show the number of source nodes generating flows, rather than destinations). The fittings performed by Keddah match closely the empirical observations. As expected, all attribute distributions can be captured using a *normal distribution* (with a variety of mean and variance parameters). This is correctly identified by Keddah's automatic model fitting.

C. Replaying Traffic

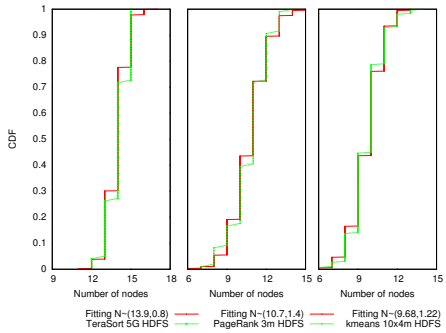
We have built an ns3 extension that can read in a Keddah file. In line with our findings, traffic is replayed in three sequential stages: HDFS Read, Shuffle, HDFS Write. The Keddah model files contain separate traffic statistics for each of these three components.

Before starting, a researcher is expected to initiate their cluster topology and other appropriate layer 2/3/4 parameters.[4] The following steps are then repeated for each of the three stages. Keddah first selects the number of source nodes that will generate traffic, and the number of destination nodes that will receive. Both are extracted from the *Number of source nodes generating flows* and the *Number of destination nodes accepting flows* attribute distributions respectively (stored within the Keddah file). These roles are then randomly allocated to nodes in the simulation. The next step is to select the number of flows that will be generated within the stage of the simulation, taken from the *Number of flows generated* attribute distribution. The size of each flow is computed by selecting values from the *Aggregated traffic volume* and *Number of flows generated* distributions, and dividing one by the other. These flows are then allocated to sources and destination pairs in a round robin fashion. Finally, each flow is given a start time by selecting values from the *Time distribution of flows* model. Once all flows in the particular stage have completed, Keddah moves the simulation onto the next Hadoop stage in the following order: HDFS Read, Shuffle, HDFS Write.

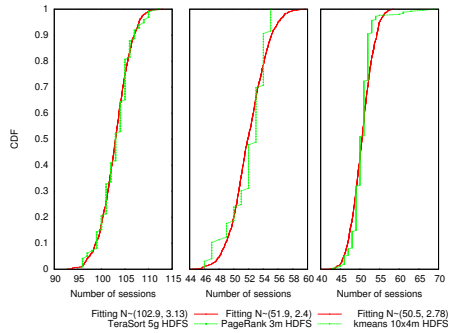
D. Validation

By design, each of the individual empirical models (e.g., time distribution, amount of traffic) closely reflects the original traffic. However, it is important to validate that the combination of these attributes to replay the flow-level traffic is also accurate. To confirm this, we use Keddah models to replay traffic in ns3 and compare it against the raw traffic seen in our testbed cluster. We build a simple simulation, which mimics our testbed

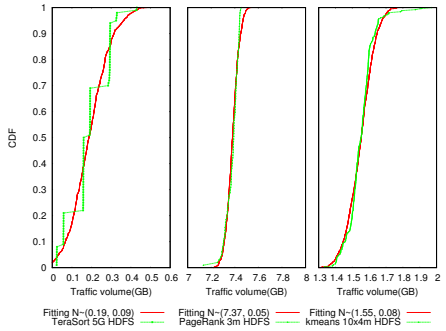
[4]Note, the simulation must contain at least the same number of nodes in the original cluster from which the Keddah model was generated.



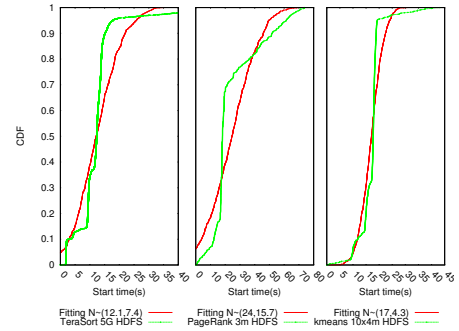
(a) Number of source nodes generating flows per run.



(b) Number of sessions per run.



(c) Aggregated traffic volume per run.



(d) Flow start time (measured by interval since job start time).

Fig. 6: Empirical model vs. fitted model for HDFS Read across TeraSort (5GB), PageRank (3m vertices) and kmeans (10x4m vertices). Each graph shows the empirical and fitted models of a different attribute.

and utilises Keddah's models to replay the traffic. We run 500 simulations, and plot again the graph shown in Figure 2. Figure 7 shows the fraction of overall traffic generated per second across both our real testbed traces and our replayed traces (for TeraSort 2GB). We observe that, visually, the trends are similar, suggesting that the replayed traffic is similar.

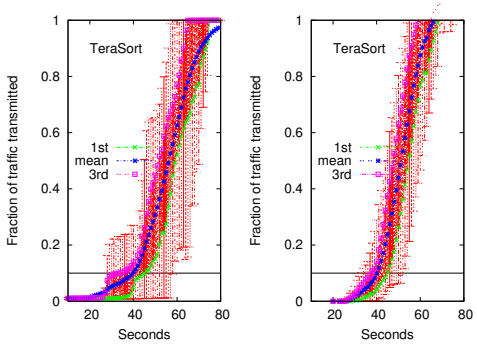


Fig. 7: Cumulative traffic generated per second in (*i*) real physical testbed traces; and (*ii*) simulated replayed traces. The horizontal line demarcates HDFS Read from Shuffle traffic as shown in Figure 2.

To quantify the similarity, we calculate the correlation between the testbed traces and the simulated traffic.

TABLE II: Correlation of cumulative traffic generated between simulation with real trace in terms of 1st quartile, mean and 3rd quartile.

Job	1st quartile	mean	3rd quartile
TeraSort 1GB	0.91	0.94	0.93
TeraSort 2GB	0.59	0.98	0.97
TeraSort 3GB	0.96	0.92	0.81
kmeans 10x5m	0.89	0.92	0.87
kmeans 10x8m	0.65	0.78	0.79

We calculate the correlations across the 1st quartile, mean and 3rd quartile of the accumulated percentage of traffic delivery shown in Figure 7. Table II presents the correlations for several job types. We observe that the correlations are generally high, confirming that the 5 attributes, indeed, allow the traffic to be replayed.

VI. Conclusion

This paper has characterised MapReduce's network traffic across several job types, decomposing their traffic components to explore their individual characteristics. To exploit this, we have built Keddah, an open source toolchain[1] that can capture and reproduce Hadoop traffic for later simulation. To date, we have used Keddah to perform a variety of experiments such as benchmarking

cluster topologies, measuring TCP incast and exploring active queue management (see our report [24]).

While our methodology has been tested exclusively in MapReduce jobs, a similar approach could be extended for in-memory distributed computing paradigms, such as Pregel [25], or Spark RDD transformations [26]. These computation models significantly improve network efficiency for iterative and multi-stage computations, but their base primitives can still be expressed in terms of MapReduce transformations. In both cases, the job execution can be expressed as a directed graph with a set of stages, whose traffic would be captured in an analogous manner to what is described in this paper.

Our future work will focus on deploying Keddah in additional environments to validate how well diverse Hadoop workloads can be captured and reproduced. We are also exploring techniques to extrapolate traffic patterns so that fewer jobs need to be executed in order to characterise the traffic of the job. Our long-term vision is that Hadoop operators may use Keddah to make their traffic models available to researchers, so that further innovations can be evaluated in realistic scenarios.

REFERENCES

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

[2] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: a survey," *ACM SIGMOD Record*, vol. 40, no. 4, pp. 11–20, 2012.

[3] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues dont matter when you can jump them!" in *Proc. ACM NSDI*, 2015.

[4] S. Narayan, S. Bailey, M. Greenway, R. Grossman, A. Heath, R. Powell, and A. Daga, "Openflow enabled hadoop over local and wide area clusters," in *High Performance Computing, Networking, Storage and Analysis (SCC)*, 2012.

[5] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating mapreduce performance using workload suites," in *Proc. IEEE MASCOTS*, 2011.

[6] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proc. ACM SoCC*, 2013.

[7] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, B.-G. Chun, and V. ICSI, "Making sense of performance in data analytics frameworks," in *Proc. ACM NSDI*, 2015.

[8] F. McSherry, "The impact of fast networks on graph analytics ," http://www.cl.cam.ac.uk/research/srg/netos/camsas/blog/2015-07-08-timely-pagerank-part1.html.

[9] L. Mai, L. Rupprecht, A. Alim, P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "NetAgg: Using middleboxes for application-specific on-path aggregation in data centres," in *Proc. ACM CoNEXT*, 2014.

[10] O. OMalley, "Terabyte sort on apache hadoop," *Available online at: http://sortbenchmark.org/Yahoo-Hadoop.pdf*, 2008.

[11] R. Nambiar, *Benchmarking Big Data Systems: Introducing TPC Express Benchmark HS*, 2015.

[12] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proc. ACM SIGMOD*, 2009.

[13] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the mapreduce-based data analysis," in *Proc. IEEE Data Engineering Workshop*, 2010.

[14] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Proc. IEEE ICDM*, 2009.

[15] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu, "Mrsim: A discrete event based mapreduce simulator," in *Proc. IEEE FSKD*, 2010.

[16] H. Yang, Z. Luan, W. Li, and D. Qian, "Mapreduce workload modeling with statistical approach," *Journal of Grid Computing*, vol. 10, no. 2, pp. 279–310, 2012.

[17] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in MapReduce setups," in *Proc. IEEE MASCOTS*, 2009.

[18] J. Tan, X. Meng, and L. Zhang, "Performance analysis of coupling scheduler for mapreduce/hadoop," in *Proc. IEEE INFOCOM*, 2012.

[19] Y. Peng, K. Chen, G. Wang, W. Bai, Z. Ma, and L. Gu, "Hadoopwatch: A first step towards comprehensive traffic forecasting in cloud computing," in *Proc. IEEE INFOCOM*, 2014.

[20] "The Apache Mahout project," http://mahout.apache.org/.

[21] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *Proc. IPDPSW*. IEEE, 2010.

[22] Z. Guo, G. Fox, and M. Zhou, "Investigation of data locality in mapreduce," in *Proc. of the 12th IEEE/ACM CCGRID*. IEEE Computer Society, 2012, pp. 419–426.

[23] 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," in *Proc. ACM EuroSys*. ACM, 2010.

[24] J. Deng, F. Cuadrado, G. Tyson, and S. Uhlig, "Hadoop network experiment use cases," https://github.com/deng113jie/keddah/blob/master/examples-append.pdf, 2017.

[25] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proc. ACM SIGMOD*, 2010.

[26] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. USENIX NSDI*, 2012.