Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015)
Krakow, Poland

Jesus Carretero, Javier Garcia Blas
Roman Wyrzykowski, Emmanuel Jeannot.
(Editors)

September 10-11, 2015

A Scheduler for Cloud Bursting of Map-Intensive Traffic Analysis Jobs

Ricardo Morla†, Pedro Gonçalves†, Jorge Barbosa‡

†INESC TEC and Faculty of Engineering, University of Porto
‡LIACC and Faculty of Engineering, University of Porto
Porto, Portugal
ricardo.morla@fe.up.pt

Abstract

Network traffic analysis is important for detecting intrusions and managing application traffic. Low cost, cluster-based traffic analysis solutions have been proposed for bulk processing of large blocks of traffic captures, scaling out the processing capability of a single network analysis node. Because of traffic intensity variations owing to the natural burstiness of network traffic, a network analysis cluster may have to be severely over-dimensioned to support 24/7 continuous packet block capture and processing. Bursting the analysis of some of the packet blocks to the cloud may attenuate the need for over-dimensioning the local cluster. In fact, existing solutions for network traffic analysis in the cloud are already providing the traditional benefits of cloud-based services to network traffic analysts and opening the door to cloud-based Elastic MapReduce-style traffic analysis solutions. In this paper we propose a scheduler of packet block network analysis jobs that chooses between sending the job to a local cluster versus sending it to a network analysis service on the cloud. We focus on map-intensive jobs such as string matching-based virus and malware detection. We present an architecture for an Hadoop-based network analysis solution including our scheduler, report on using this approach in a small cluster, and show scheduling performance results obtained through simulation. We achieve up to more than 50% reduction on the amount of network traffic we need to burst out using our scheduler compared to simple traffic threshold scheduler and full resource availability scheduler. Finally we discuss scaling out issues for our network analysis solution.

Keywords Packet Network Traffic Analysis, Hadoop, Cloud Bursting

I. Introduction

Many companies invest in their private IT data centers to meet most of their needs. However, these private data centers might not cope with workload peaks, leading to delays and lower throughput. Using extra computing power to handle the unpredictable and infrequent peak data workloads is expensive and inefficient, since a portion of the resources will be inactive most of the time. Migrating the whole application to a cloud infrastructure, though possibly cheaper than investing in the private data center because servers are only rented when needed, is still expensive and could compromise private and important data. A hybrid model offers the best solution to address the needs for elasticity when peak workloads appear, while providing local privacy if needed. In this model, the local IT data center of an enterprise is optimized to fulfill the vast majority of its needs, resorting to additional resources in the cloud when the local data center resources become scarce [1]. This technique is called Cloud Bursting and enables an enterprise to scale out their local infrastructure by bursting their applications to a third-party public cloud seamlessly, if the need arises [2].

Cloud Bursting could be of use to network managers and security experts. In fact, their need for more computational power that can perform more

complex network intrusion detection and application traffic management is hand in hand with the variability of network traffic and of traffic analysis workload. Big data style analysis software and services using the open source YARN and Hadoop platform[1] are available both in the research community [3, 4] and commercially[2]. Although [3] in particular provides a range of network analysis jobs from low level packet and flow statistics to intrusion detection, their performance evaluation focuses mainly on the throughput of the analysis system with different file sizes. In particular, Map and Reduce run time distributions that could be helpful for large scale evaluation have not been characterized.

Three types of network traffic analysis are typical [5]: 1) real-time analysis, where continuous streams of data are analyzed and processed as they arrive; 2) batched analysis, where data is aggregated in batches and analyzed periodically; and 3) forensics analysis are analysis that only occur when special events are triggered, for example, when a major intrusion has been detected and requires detailed analysis of logged traffic. This work focuses on 24/7 continuous batch analyses of consecutive captures of network packet traffic and on the natural variability that characterizes such traffic. Because intrusion detection results for each batch must be delivered as soon as possible to allow for early detection, traffic variability in a continuous batch processing means either over-dimensioning the computing cluster or waiting longer to get detection results. Our proposal is to use Cloud Bursting to burst some jobs and achieve lower waiting times. We present an integrated solution in section II that we have implemented in our networking laboratory. An important part of this solution is the scheduler that decides whether to burst a job based on the size of the file that needs to be analyzed and on the resource usage in the cluster. We present our Map-intensive job, job model, scheduler rationale, and cluster capacity estimation in section III. In section IV we show results of running our network analysis job in a small Hadoop cluster. The goal here is to provide an example of how the scheduler works and to obtain an empirical distribution for the Map run time to be used in our simulator in section V. In section V we describe the simulator, provide a distribution for link load, and characterize run times of single and continuously arriving jobs. In section VI we show job delay and burst count for our scheduler and compare it with baseline schedulers. We conclude with scale out analysis in section VII, related work analysis in section VIII, and final remarks in section IX.

[1] http://hadoop.apache.org
[2] http://www.pravail.com

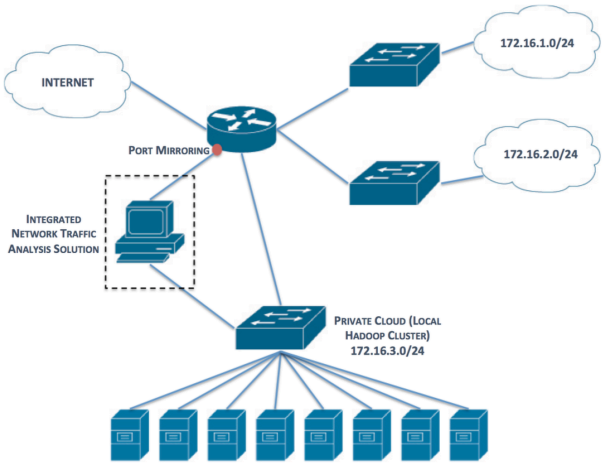

Figure 1: Our network topology with port mirroring, PCAP traffic file capture, cluster for PCAP file processing, and our control node (INTAS).

II. Cloud Bursting Architecture

Figure 1 shows the topology of our network. Outbound traffic from the 172.16.1.0/24 and 172.16.2.0/24 networks is captured via port mirroring and the tcpdump application in our Integrated Network Traffic Analysis Solution (INTAS). INTAS will decide whether to send each PCAP file created by tcpdump to our local private cloud or to burst the file to a public cloud, reachable through the Internet. In this paper we assume the public cloud is provisioned such that an adequate job performance is achieved.

Figure 2 shows the architecture and modules of our solution. The Network Traffic Gatherer module uses tcpdump to capture batches of network traffic from a pre-specified network interface and generate PCAP files. Once ready, the PCAP files are copied to a folder

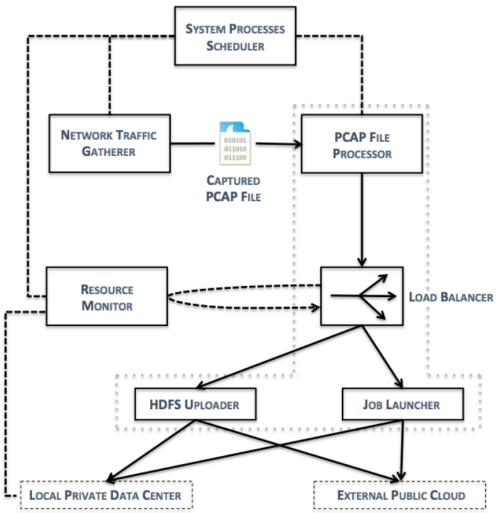

Figure 2: The architecture of INTAS.

where the PCAP File Processor module determines whether or not the PCAP file is new before delivering it to the Scheduler/Load Balancer. Using the cluster resource utilization information provided by the Resource Monitor, the Scheduler decides whether to launch the job locally or to burst it. In either case, the PCAP file is first uploaded to HDFS on the destination cluster through the HDFS Uploader module. After the upload is complete, the Job Launcher connects to the Hadoop cluster to launch the map-intensive network analysis job.

III. Scheduler

III.1 Map-Intensive Network Analysis Jobs

Map-intensive network analysis jobs such as traffic classification and virus or malware detection using string matching can have the following two-phase design. The Map phase checks packet payload for the presence of signature keys. This outputs a list of which applications or viruses were identified on which packets. The reduce phase merges these results and writes them to the job's output file.

The Map phase of this kind of network analysis job is computationally expensive. Simply put, all possible string sequences in the packet's transport layer payload must be compared to the signature keys of the different viruses, malware, or specific applications we want to find. For a signature key with b_{key} bytes and a string comparison factor $k^{key}[second/byte]$, the total processing time of a packet is $t_{packet} = k^{key}(b_{packet} - b_{key})$, with $b_{packet} \geq b_{key}$. For small keys and relatively large packets, we can have a close approximation of t_{packet} by using its upper bound $k\,b_{packet}$, where k is the sum of all string comparison factors and which can also be expressed as $k = nk_{avg}$ with n keys and k_{avg} average string comparison factor. This upper bound is proportional to the size of the packet. We find an estimate of k by benchmarking the target cluster as reported later in the paper.

Under the Map-Reduce paradigm, each traffic capture file is split into a number of data blocks and each block is processed by a Map process. Although the exact number of packets in each block can change according to the nature of the network traffic, the size in bytes of almost all data blocks is pre-configured and independent of the nature of the traffic. Our estimate for the Map processing time of a block with b_{block} bytes is $t_{block} = k\,b_{block}$. We assume the switching time between processing of packets is much smaller than the actual processing.

The Reduce phase of this kind of network analysis jobs is much less computationally demanding than the Map phase. We assume the Reduce processing time to be zero.

III.2 Job Completion Time Estimate

The number of map tasks that a network analysis job launches depends on the size of the input traffic capture file and on the Map-Reduce data block size. A block size of 128 MB is typical, which for a slightly under 900 MB traffic capture file size b_{file} would yield 7 data blocks on which 7 Map processes would need to run. More generically, we compute $m = b_{file}/b_{block}$ and estimate the number of Map processes to be $M = \lceil m \rceil$. We disable speculative execution to improve this estimate.

The container is the unit of resource allocation in Hadoop YARN. Typically, two containers run per host.

The mapping to containers sets the limit to the number of Maps that a network analysis task can run simultaneously. One Map process runs on a container. Each Map-Reduce job is managed by an ApplicationMaster that requires an additional host throughout the complete lifetime of the job. Although Reduces also require containers, their impact on resource usage in this kind of network analysis job is much smaller than that of Maps and we do not consider them.

For a YARN cluster with H hosts, the number of Maps that can run simultaneously is $M_s = 2(H-1) - 1$. Comparing M_s and M takes us to the concept of Map waves. Each wave of M_s Maps will run simultaneously for approximately t_{block}, after which a new wave of M_s (or fewer) Maps will start. This repeats until all M Maps have run. We compute $n = M/M_s$ and estimate the number of waves as $N = \lceil n \rceil$. Our estimate for the job completion time is $t_{Mjob} = (N-1) * t_{block} + k(m - \lfloor m \rfloor)b_{block}$ if the last wave has a single Map task (in which case $M - NM_s = 1$), and $t_{Mjob} = N * t_{block}$ otherwise, which is the more frequent case.

In addition to T_{Mjob}, the scheduler needs have an estimate of the time it takes the system to upload the captured traffic file from the capture node to HDFS. We use a simple estimate $T_{upload} = b_{file}/r$ with r in $[bytes/second]$. The total job completion time estimate is $T_{job} = T_{Mjob} + T_{upload}$.

At any moment in time after the job has been scheduled, the estimate for the job completion time $T_{job}^{remaining}$ can be updated as follows:

- $T_{Mjob} + (b_{file} - b_{file}^{uploaded})/r$, if the upload not finished yet. $b_{file}^{uploaded}$ is the number of bytes that have been uploaded so far.
- $T_{Mjob} - N^{done}\, t_{block} - t_{wave}$, if the upload finished. N^{done} is the number of completed waves and t_{wave} is how long the current wave has been running.

This estimate is valid for a single job running in the cluster.

III.3 Scheduling Concurrent Jobs

An incoming traffic analysis job is scheduled regardless of the size of the capture file when no job is running on the cluster. When traffic peaks, it is possible that a new chunk of traffic is captured and ready for analysis before the traffic analysis job of the previous chunk is over. In this case where a job is running in the local cluster when a new job arrives, the scheduler will have to decide whether to send the incoming job to the local cluster or to the cloud.

Our baseline scheduler verifies if there are enough containers for the job in the local cluster. If there are, the job is ran locally; otherwise the job is sent to the public cloud. We assume the connection to the public cloud and the public cloud cluster itself are provisioned such that an adequate response time is achieved. We do not explore public cloud performance in this paper.

The approach of the baseline scheduler is over simplifying and can send more jobs to the public cloud than needed: 1) the time it takes to upload the capture file to the local HDFS can be enough for the current job to complete and the incoming job to be processed locally; 2) the last wave of a job may use fewer resources than the previous waves, which can be used by the incoming job. Our proposed scheduler takes advantage of the job completion time estimate and wave model presented in the previous section to try to reduce the number of uploads to the public cloud and reducing the impact on job completion time.

Our proposed scheduler schedules incoming candidate job locally if:

- **S0.** Local cluster is empty or enough containers are available to run the incoming job.
- **S1.** Local cluster has single current job and current job finishes before or up to Th_1 seconds after incoming job upload time to local cluster HDFS. For this we use $T_{job}^{remaining}$ of the current job and T_{upload} of the incoming candidate job. Th_1 provides some tolerance to the decision and can be chosen to be a few percent of the wave duration.
- **S2.** Local cluster has single current job and: 1) the next to the last wave of current job finishes before or up to Th_1 seconds after incoming job upload time to local cluster HDFS and 2) the number of containers of the last wave of the candidate incoming job is smaller than or equal to the number of

containers not used by last wave of the current job.

III.4 Benchmarking and Estimating Capacity

Our scheduler requires an estimate of the block execution time t_{block} and of the upload to HDFS rate r. Our approach for estimating these values is to run the network analysis job we are interested in benchmarking on an sample data set prior to running the job on the target traffic capture files. This provides samples for measure Map completion time and file upload time, which can be averaged or maxed and used as estimate for t_{block} and r.

Estimating the capacity of a cluster for this kind of network analysis is important for: 1) defining the traffic throughput that a given cluster can support for analysis and 2) specifying how many and what kind of nodes there should be in the cluster for a given traffic throughput that needs to be analyzed. We estimate the maximum throughput of a traffic capture that a cluster can support as $T_{hput} = M_s * b_{block}/t_{block} = (2(H-1)-1)/k$. Using this estimate, a cluster with 10 nodes, 128 MB block size, and $t_{block} = 5min\,(k = 2.23\,\mu s/byte)$ would support approximately 60 Mbit/s traffic captures, which yields a maximum 10 minute traffic capture file of 4.5 GB.

IV. String Matching Job in Small Physical Clusters

We ran a Map-intensive string matching job with 200 signature keys on H=5 node low-end YARN clusters. To more easily launch our Hadoop 2.3.0 cluster we use a private cloud based on Openstack IceHouse[3] and the Sahara Openstack module[4] that provisions data-intensive application clusters like YARN. The bare-metal operating system is Ubuntu 14.04. Each bare metal runs a single virtual machine YARN node.

Due to time and lab resource constraints we ran the two experiments of this section in different hardware. The scheduling experiment was run on a heterogeneous cluster with the following bare-metals: Intel Core i7-3770 3.40GHz, two Intel Core i7 950 3.07GHz, Intel E5504 2.00GHz. The map runtime experiment was run on a homogeneous cluster in which the bare-metal CPUs are older, 2008 Intel Core 2 Quad Q9300 running at 2.50GHz. CPU names are as reported by /proc/cpuinfo.

We captured two PCAP files locally on a 100 Mbit/s link for 10 minutes each. One file (A) is 1.22 GB and the other (B) is 2.01 GB, corresponding to 17 Mbit/s and 26 Mbit/s of traffic going out of our local network. One file has 10 128 MB data blocks and the other 16.

IV.1 Scheduling

For our scheduling experiments we replayed the two PCAP files A and B into our system every 10 minutes according to this sequence: "ABBBBA-ABABAA-AAAAAA". File A has the cluster running slightly below capacity whereas file B has the cluster running above capacity. This means that during the first phase "ABBBBA" the cluster will be running well above capacity, during the second phase "ABABAA" the cluster will be just slightly over capacity, and during the third phase "AAAAAA" the cluster will be running below capacity.

Figures 3 and 4 show an example of processing the first phase "ABBBBA". To produce analysis results as soon as possible, the Maps for job 6A should start immediately after the upload is completed, i.e. on the right side of the small box near 6A. Notice that without scheduler (figure 3, job id 54), the Maps have to wait until the previous job completes – which is a considerable amount of time. In figure 4 with the scheduler and by bursting job 5B, job 6A can start immediately after upload.

Figure 5 shows how the job sequence was burst or delayed using the three different scheduling approaches: no scheduler, simple, and advanced. Because we want to have an idea of the impact of the scheduler on each job individually, we can compare the run time of a job T_{job} with the sum of the job's Map run times divided by the number M_s of Maps that can run simultaneously. We define this metric as $D = T_{job}/T_{ideal} - 1$ with $T_{ideal} = 1/M_s \sum_i t^i_{block}$ and where t^i_{block} is the run time of the i^{th} Map of the job. The results confirm our intuition that by bursting some jobs the job delay can

[3]https://www.openstack.org/software/icehouse/
[4]http://docs.openstack.org/developer/sahara/

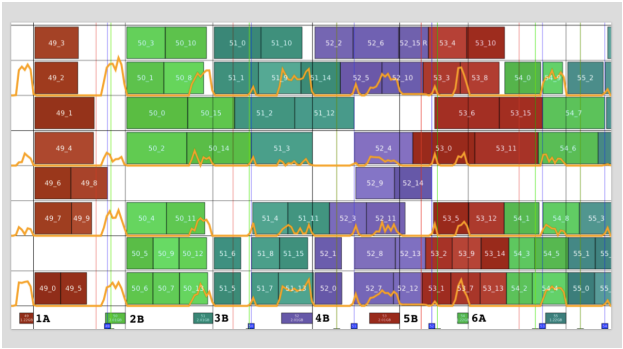

Figure 3: Example of processing phase "ABBBBA" without bursting. Horizontal lines are the placeholder for container utilization through time. "X_Y" labeled boxes represent Map run Y of job X. Smaller boxes with file sizes and e.g. "1A" legend represent PCAP file upload to HDFS. Plot lines on each pair of containers represent traffic in and out of the network interface in the bare metal where the containers run.

be reduced. The advanced scheduler only burst 3 jobs while not causing distinguishable delays compared to the simple scheduler that burst 5 jobs. Interestingly, the results for job delay without scheduler suggest some spillover from phase 1 to phase 2 that seems to make job delay in phase 2 larger than in phase 1 when in fact phase 1 is more demanding than phase 2. This could also be due to the inherent randomness of this system. This is one of the reasons why we now go to simulation and better understand the performance of the bursting approaches.

IV.2 Map run time distribution

Figure 6 shows the distribution of the run time and throughput of 762 128MB-block Maps that were used to analyze files A and B. We ran the analysis job on file A 48 times and on file B 24 times. The Map run time 50^{th} percentile is 6 minutes and 3 seconds. We can use this value to estimate the capacity of our cluster as follows: $k = 363s/128MB = 2.70\mu s/byte$ and thus $T_{hput} = 20.7Mbit/s$. With 10 minute captures this yields a file size of 1.44 GB. We use this value in our scheduler. A more conservative approach would choose the 90^{th} percentile and a resource utilization factor of 2/3 yielding $k = 494s/128MB/\frac{2}{3} = 5.52\mu s/byte$ and $T_{hput} = 10.64Mbit/s$.

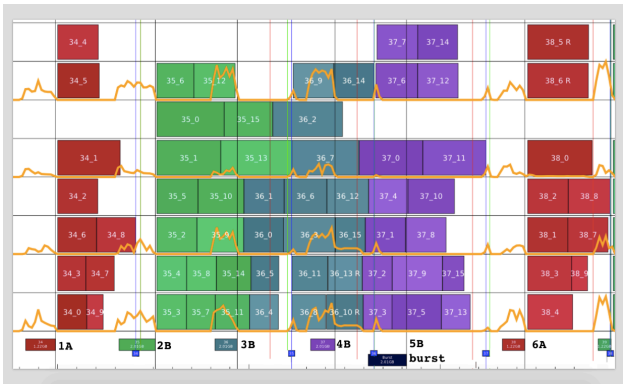

Figure 4: Example of processing phase "ABBBBA" with bursting.

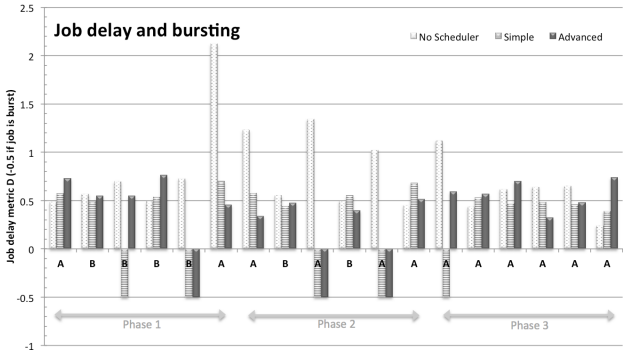

Figure 5: Job delay and bursting for different scheduling options on the example sequence of network analysis jobs.

V. Simulator

V.1 Design

We built an event-driven simulator in python to simulate map-intensive traffic analysis jobs. The simulator keeps a list of jobs to be scheduled *job_launch_schedule*, including their randomly generated file sizes and the times at which they are be launched in the cluster. This corresponds at the times at which a PCAP file is ready to be processed in the YARN cluster. For this paper it's every 10 minutes.

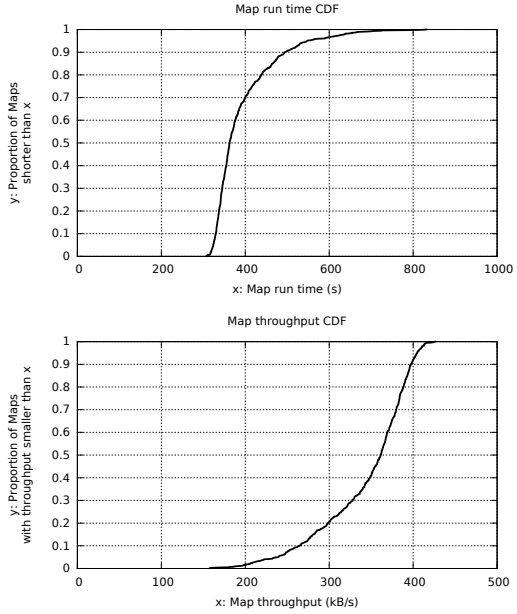

Figure 6: Empirical CDFs of run time and throughput of 762 Maps of our network analysis job.

This list is populated before the simulation starts and jobs are popped out and launched as the simulation progresses. YARN can be configured to allow a maximum of simultaneously running applications in the cluster. To account for this, a job that is popped out of the *job_launch_schedule* list will go to a *job_waiting* list. If the number of currently running applications is below the maximum, the oldest job is removed from the *job_waiting* list and included in the *job_running* list. Jobs in the *job_runnig* list will compete for available cluster containers in round-robin. Only the job at the top of the list will get a container to run a Map. For every container a running job gets to run a Map it will be pushed to the back of the list. Container release events are scheduled according to the randomly generated Map run time distribution and processed by the simulation together with job launch events. The simulation finishes when there are no more events to process or maximum simulation time is reached. Block size is 128MB.

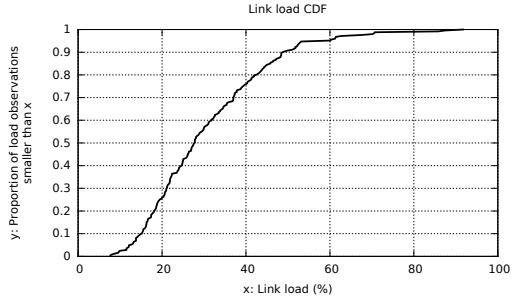

Figure 7: 10 Gbps link load empirical CDF.

V.2 Workload Distributions

Synthetic workload generation requires two components. The first is a distribution for the Map run time, for which we use the Map run time empirical distribution obtained from our 5 node clusters and shown in figure 6. The second is the PCAP file size distribution that determines the number of data blocks and Map tasks that the job needs to process. As this is directly related to the amount of traffic through a packet network, we build on publicly available data from the CAIDA Center for Applied Internet Data Analysis in San Diego, CA[5]. We use every month, hour-long average bitrate on both directions of their 10 Gbps San Diego links to Chicago and San Jose to build a 247 point empirical link load distribution. We show the CDF of this distribution in figure 7. Average link load is 31%.

Figure 8 shows the Q-Q plots for our synthetic data and the empirical distributions. Notice how close the Q-Q points are to the $y = x$ line. Because we have fewer observations in the high link load region the points there are not as close to the $y = x$ line as the other points.

Simulation results in the rest of the paper are obtained by running 10^4 jobs for each link capacity. In continuously arriving jobs including when the schedulers are used we run each set of 10^4 jobs 20 times to get final results.

[5] The CAIDA UCSD http://www.caida.org/data/passive/trace_stats/

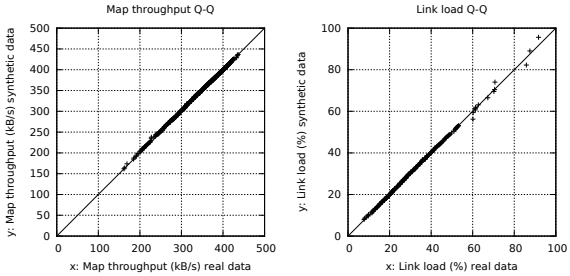

Figure 8: Q-Q plots comparing 10k synthetic data points and the empirical distributions of Map throughput and link load from figures 6 and 7.

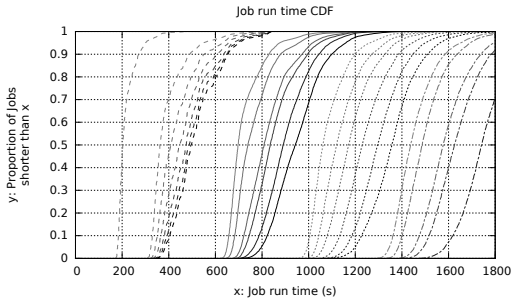

Figure 9: CDFs of the run time of fixed workload jobs on an $M_s = 7$ cluster. From left to right the link throughput values in Mbit/s are: (1, 2, 5, 7, 10, 12, 12.53), (15, 17, 20, 22, 24.5, 25.05), (27, 30, 32, 35, 37, 37.58), (40, 42, 45, 47, 50, 100). The 100 Mbit/s line is out of range. The CDFs for each link throughput form groups according to the number of waves that our model indicates (1:long dash, 2: solid, 3:short dash, 4:long-short dash). Line width in each group increases with link throughput.

V.3 Single Job

Understanding the performance of a cluster under a single job workload helps validate the simulator and can provide insights into the performance analysis of our scheduler under a continuous workload. Figure 9 shows the run time CDF of a single job for a fixed link bitrate that we vary from 1 Mbps to 40 Mbps on our H=5 simulated cluster. The bitrate grows from left CDF to right CDF. We can notice groups of CDFs that correspond to waves in our job model. The first group goes up to 12.53 Mbit/s which yields a 7 128MB block PCAP file requiring exactly the number of Maps (7) that can run simultaneously in a wave. We call this the wave boundary bit rate. With a slight increase to 15 Mbit/s, file size of 1073 MB, and 9 Maps (8 128 blocks and 1 48 MB block), there is a significant increase in the 50^{th} percentile run time from 495 s at 12.53 Mbit/s to 698 s at 15 Mbit/s, as this file size requires two waves of Maps. As the link traffic increases to include multiple waves this difference seems to decrease. Unless stated otherwise, link throughput values and their linestyle mapping in figure 9 are used throughout the paper.

As discussed in section V.2, link throughput is not constant and this has an impact on job runtime and resource usage CDFs. Figure 10 shows run time for links with capacity equal to the fixed bitrate values used in the previous section. Run times are generally smaller than those in figure 9 as the random workload distribution is applied to the link capacity and on average results in smaller bitrates and smaller PCAP files to process. Notice that the wave phenomenon is no longer obvious to observe except for the first two link capacity values.

V.4 Continuously Arriving Jobs

We now consider the case that we are most interested in: PCAP files freshly captured and uploaded to be processed by our network analysis job every 10 minutes. Bit rate and Map processing time distributions will make some jobs last more than 10 minutes, in

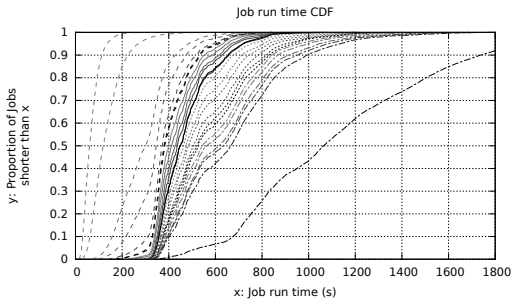

Figure 10: CDFs of the run time of random workload jobs on an $M_s = 7$ cluster following our random workload distribution.

which case the next job will start later. Because we are interested in obtaining the results of the analysis as soon as possible after PCAP file capture and upload to the cluster, we include this delay in the job run time. Job run time results for this case are shown in figure 11 and the Q-Q plot comparison with the single job case is shown in figure 12. The effect of the current job delaying the next is not noticeable in the lower capacity links. Above 27 Mbit/s the Q-Q plots get noticeably away from the $y = x$ line, especially for job run times higher than 400 s. This means that with these link capacities a significant number of jobs will be starting to experience delay because the previous job did not finish on time. For the 100 Mbit/s link the job run time will eventually increase monotonically with each incoming job at which point the cluster is unable to keep up. For the range of link capacities that we are studying in our $M_s = 7$ cluster, our bursting approach and scheduler will likely only be useful above 27 and below 100 Mbit/s where congestion exists but is moderate.

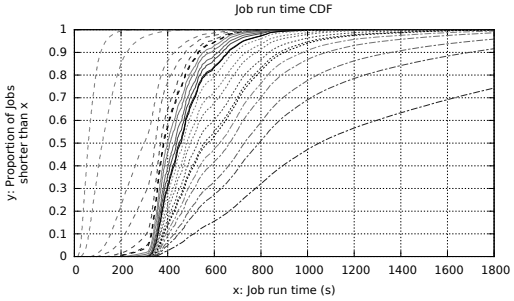

Figure 11: CDFs of the run time of random workload jobs on an $M_s = 7$ cluster following our random workload distribution but now with continuously arriving jobs. The 100 Mbit/s line is out of range.

VI. Scheduler Performance Results

In this section we compare four approaches: no scheduler, a simple scheduler using S_0 from section III.3, our proposed advanced scheduler using S_0, S_1, and S_2, and a traffic threshold-based scheduler that bursts all jobs with more than 50% link load. We compare these approaches on two aspects for each link bandwidth: 1) Q-Q job run time distributions with respect to singe

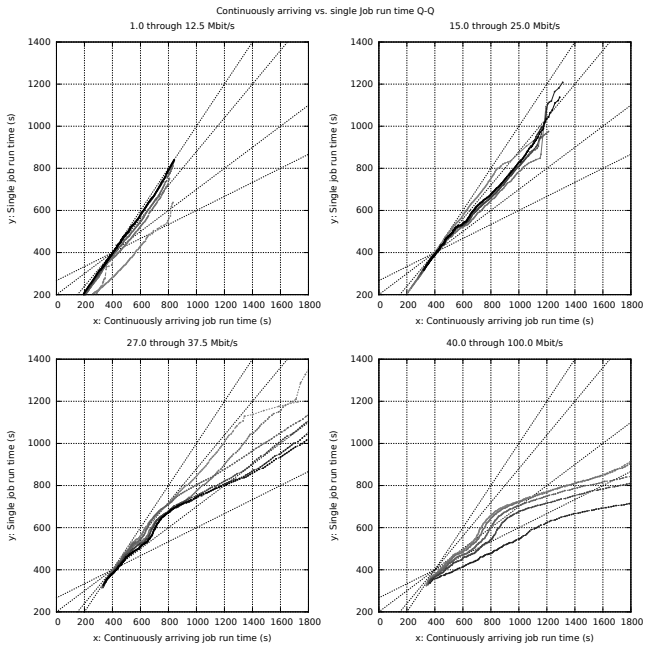

Figure 12: Q-Q plot comparing the job run time distributions of each link capacity with a single job (y) and continuously arriving jobs (x).

job as in figure 12 and 2) proportion of traffic that is burst. We include the run time of burst traffic in our Q-Q analysis by considering that it will be processed as a single job on the cloud. Noting that the Q-Q plots for the simple, advanced, and threshold approaches are visually similar to each other and to that of the approach without scheduler shown in figure 12, we use the following metric T to quantify their differences. Consider y and x as the Q-Q run time values for the run time distribution of single jobs (y) and the run time distribution for a given scheduler approach (x). We define $t = y/x - 1$, that is negative for each point where the run time value of the scheduler approach is larger than that of the single job. The scheduler performance metric we define is $T = 1/n \sum t$ for an n point Q-Q plot. Figure 13 shows metric T and the proportion of burst traffic for our set of link capacities. The four approaches have similar run times just slightly worse than single job up to 25 Mbit/s link capacity. From that value the run time T metric of the approach without scheduler degrades progressively whereas the

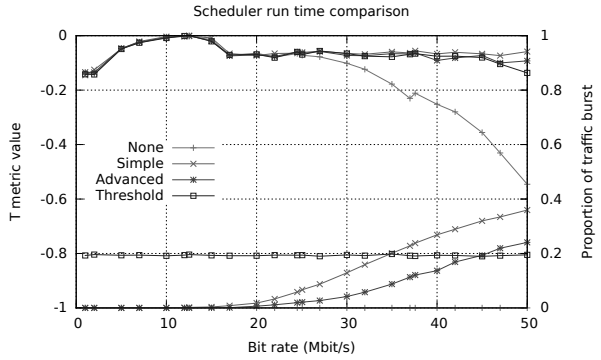

Figure 13: T metric and proportion of traffic burst for different scheduler approaches. T metric values are closer to the top of the graph, traffic burst values closer to the bottom.

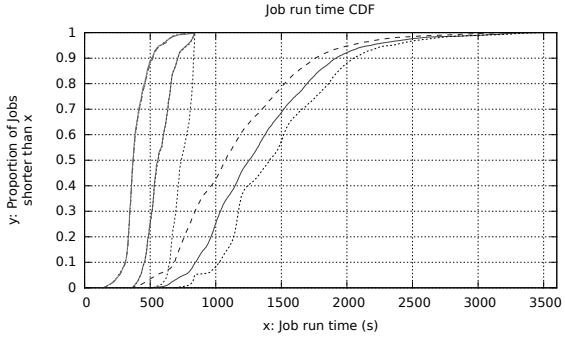

Figure 14: CDFs of the run time of random workload jobs for typical link bitrates on clusters of different sizes. Long dash: 10 Mbit/s and 100 Mbit/s on an M_s=7 cluster. Solid line: 10 Mbit/s, 100 Mbit/s, and 1 Gbit/s on an M_s = 70 cluster. Short dash: 10 Mbit/s, 100 Mbit/s, 1 Gbit/s, and 10 Gbit/s on an M_s = 700 cluster.

other three approaches are reasonably similar and not much worse than lower link capacities or single job. Looking at burst traffic, the simple approach bursts almost twice as much traffic as the advanced approach. The approach with less burst traffic is the advanced scheduler up to 45 Mbit/s. At 35 Mbit/s the advanced scheduler bursts 56% less traffic than the other two schedulers. Thus for the range of values where bursting could be useful, i.e. from 27 Mbit/s and below 100 Mbit/s, the advanced scheduler yields both the smallest amount of burst traffic and single-job comparable run times.

VII. Scaling Out

12 Mbit/s and the other bit rate values used in figure 9 are not link capacities that can be found in typical network links. Four typical link capacities are 10 Mbit/s, 100 Mbit/s, 1Gbit/s, and 10Gbit/s. The conservative capacity estimation approach in section IV.2 puts our $M_s = 7$ cluster at approximately 11 Mbit/s, which is not enough for the four typical link capacities. With that in mind, in figure 14 we show job run time for the four typical link capacities on clusters of 3 different sizes: our $M_s = 7$ initial cluster and $M_s = 70$ and $M_s = 700$ clusters.

VIII. Related Work

In the introduction we provided arguments for the relevance of our approach in the context of the analysis of network traffic. In summary, cloud bursting of batched Map-intensive network analysis jobs and the scheduling of such bursts has not been proposed before in the traffic analysis literature. In this section we take a look at different general-purpose cloud bursting approaches and argue that they are not adequate to our batched Map-intensive network analysis jobs.

Existing open-source virtualized data center management tools such as OpenStack and OpenNebula already support cloud bursting. Their initial focus was to provide an abstraction layer for the low level details of transitioning VMs (Virtual Machines) between data centers [1]. Throughout recent years, continuous improvements have been made to the amount of provided features and configurable parameters to better suit the users' needs. Today, many available solutions including Seagull [1] and the OPTIMIS Project [2] offer scheduling policies that determine if and when to burst to the cloud. However, these solutions are typically geared towards web applications and do not adequately support applications that need to process large amounts of data.

As the world embraces the ever-growing paradigm

of Big Data, Cloud Bursting can also be used in the context where cloud resources are used to store additional data if local resources become scarce. In fact, the use of Cloud Bursting for data-intensive computing has not only been proved feasible but scalable as well. [6] presents a middleware that supports a custom MapReduce type API and enables data-intensive computing with Cloud Bursting in a scenario where the data is split across a local private data center and a public cloud data center. Data is processed using computing power from both the local and public cloud data centers. Furthermore, data on one end can be processed with computing power from the other end, albeit lowering the overall job execution time. BStream [7] is a Cloud Bursting implementation that uses Hadoop YARN with MapReduce in the local data center and the Storm stream processing engine in the cloud to process MapReduce jobs instead of using YARN. The use of Storm allows the output of Maps to be streamed to Reduces in the cloud. Both [6] and [7] are better suited for forensic jobs for which a large data set must be analyzed and outputs from local Maps need to be processed on the cloud, than for batched analysis of smaller yet still computationally demanding data sets. In our case jobs can fit both in the local data center or the public cloud and the challenge is to process continuously arriving jobs.

IX. Conclusion

We have set out to explore cloud bursting for Map-intensive network traffic analysis jobs. Using our proposed architecture for collecting, cloud bursting, and processing traffic on Hadoop clusters, we characterized the run times of Maps on our physical clusters and used it to drive a simulation assessment of our job model and cloud bursting scheduler. Our scheduler bursts out up to more than 50% less traffic than other schedulers we compared. We plan to extend the traffic analysis modeling to other types of network traffic analyses and on platforms such as Spark using GPUs and to understand heterogeneity and energy consumption issues of scaling out the traffic analysis.

References

[1] T. Guo, U. Sharma, P. Shenoy, T. Wood, and S. Sahu. *Cost-Aware Cloud Bursting for Enterprise Applications*, ACM Transactions on Internet Technology, 13(3):1-24, 2014.

[2] S. K. Nair, et al. *Towards Secure Cloud Bursting, Brokerage and Aggregation*. Proceedings of the 8th IEEE European Conference on Web Services, ECOWS 2010, pages 189-196, 2010.

[3] Y. Lee and Y Lee. *Toward scalable internet traffic measurement and analysis with Hadoop*, ACM SIGCOMM Computer Communication Review, 43(1):5-13, 2012.

[4] RIPE. *Large-scale PCAP Data Analysis Using Apache Hadoop*, `https://github.com/RIPE-NCC/hadoop-pcap`, 2012.

[5] A. Pallavi and P. Hemlata. *Network Traffic Analysis Using Packet Sniffer*, International Journal of Engineering Research and Applications, 2(3):854-856, 2012.

[6] T. Bicer, D. Chiu, and G. Agrawal. *A Framework for Data-Intensive Computing with Cloud Bursting*. 2011 IEEE International Conference on Cluster Computing, pages 169âĂŞ177, September 2011.

[7] S. Kailasam, P. Dhawalia, S. J. Balaji, G. Iyer, and J. Dharanipragada. *Extending MapReduce across Clouds with BStream*. IEEE Transactions on Cloud Computing, 2(3):362-376, July 2014.