The Nature of Datacenter Traffic: Measurements & Analysis

Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, Ronnie Chaiken
Microsoft Research

ABSTRACT

We explore the nature of traffic in data centers, designed to support the mining of massive data sets. We instrument the servers to collect socket-level logs, with negligible performance impact. In a 1500 server operational cluster, we thus amass roughly a petabyte of measurements over two months, from which we obtain and report detailed views of traffic and congestion conditions and patterns. We further consider whether traffic matrices in the cluster might be obtained instead via tomographic inference from coarser-grained counter data.

Categories and Subject Descriptors
C.2.4 [**Distributed Systems**] Distributed applications
C.4 [**Performance of systems**] Performance Attributes
General Terms
Design, experimentation, measurement, performance
Keywords
Data center traffic, characterization, models, tomography

1. INTRODUCTION

Analysis of massive data sets is a major driver for today's data centers [6]. For example, web search relies on continuously collecting and analyzing billions of web pages to build fresh indexes and mining of click-stream data to improve search quality. As a result, distributed infrastructures that support query processing on peta-bytes of data using commodity servers are increasingly prevalent (e.g., GFS, BigTable [9, 17], Yahoo's Hadoop, PIG [4, 27] and Microsoft's Cosmos, Scope [8, 23]). Besides search providers, the economics and performance of these clusters appeals to commercial cloud computing providers who offer fee based access to such infrastructures [1, 3, 5].

To the best of our knowledge, this paper provides the first description of the characteristics of traffic arising in an operational distributed query processing cluster that supports diverse workloads created in the course of solving business and engineering problems. Our measurements collected network related events from each of the 1500 servers, which represent a logical cluster in an operational data center housing tens of thousands of servers, for over two months. Our contributions are as follows:

Measurement Instrumentation. We describe a lightweight, extensible instrumentation and analysis methodology that measures traffic on data center servers, rather than switches, providing socket level logs. This server-centric approach, we believe, provides an advantageous tradeoff for monitoring traffic in data centers. Server overhead (CPU, memory, storage) is relatively small, though the traffic volumes generated in total are large – over 10 GB per server per day. Further, such server instrumentation enables linking up network traffic to the applications that generate or depend on it, letting us understand the causes (and impact) of network incidents.

IMC'09, November 4–6, 2009, Chicago, Illinois, USA.

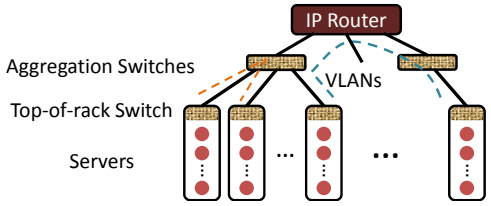

Figure 1: Sketch of a typical cluster. Tens of servers per rack are connected via inexpensive top of rack switches that in turn connect to high degree aggregation switches. VLANs are set-up between small numbers of racks to keep broadcast domains small. We collect traces from all (1500) nodes in a production cluster.

Traffic Characteristics. Much of the traffic volume could be explained by two clearly visible patterns which we call *Work-Seeks-Bandwidth* and *Scatter-Gather*. Using socket level logs, we investigate the nature of the traffic within these patterns: flow characteristics, congestion, and rate of change of the traffic mix.

Tomography Inference Accuracy. Will the familiar inference methods to obtain traffic matrices in the Internet Service Provider (ISP) networks extend to data centers [20, 32, 34, 35]? If they do, the barrier to understand the traffic characteristics of datacenters will be lowered from the detailed instrumentation that we have done here to analyzing the more easily available SNMP link counters. Our evaluation shows that tomography performs poorly for data center traffic and we postulate some reasons for this.

A consistent theme that runs through our investigation is that the methodology that works in the data center and the results seen in the data center are different than their counterparts in ISP or even enterprise networks. The opportunities and "sweet spots" for instrumentation are different. The characteristics of the traffic are different, as are the challenges of associated inference problems. Simple intuitive explanations arise from engineering considerations, where there is tighter coupling in application's use of network, computing, and storage resources, than that is seen in other settings.

2. DATA & METHODOLOGY

We briefly present our instrumentation methodology. Measurements in ISPs and enterprises concentrate on instrumenting the network devices with the following choices:

SNMP counters, which support packet and byte counts across individual switch interfaces and related metrics, are ubiquitously available on network devices. However, logistic concerns on how often routers can be polled limit availability to coarse time-scales, typically once every five minutes, and by itself SNMP provides little insight into flow-level or even host-level behavior.

Sampled flow or sampled packet header level data [16, 29, 22, 31] can provide flow level insight at the cost of keeping a higher volume of data for analysis and for assurance that samples are representative [15]. While not yet ubiquitous, these capabilities are becoming more available, especially on newer platforms [12].

Deep packet inspection: Much research mitigates the costs of packet inspection at high speed [11, 13] but few commercial devices support these across production switch and router interfaces.

In this context, how do we design data-center measurements that achieve accurate and useful data while keeping costs manageable? What drives cost is detailed measurement at very high speed. To achieve speed, the computations have to be implemented in firmware and more importantly the high speed memory or storage required to keep track of details is expensive causing little of it to be available on-board the switch or router. Datacenters provide a unique choice— rather than collecting data on network devices with limited capabilities for measurement, we could obtain measurements at the servers, even commodity versions of which have multiple cores, GBs of memory, and 100s of GBs or more of local storage. When divided across servers, the per-server monitoring task is a surprisingly small fraction of what a network device might incur. Further, modern data centers have a common management framework spanning their entire environment–servers, storage, and network, simplifying the task of managing measurements and storing the produced data. Finally, instrumentation at the servers allows us to link the network traffic with application level logs (e.g., at the level of individual processes) which is otherwise impossible to do with reasonable accuracy. This lets us understand not only the origins of network traffic but also the impact of network incidents (such as congestion, incast) on applications.

The idea of using servers to ease operations is not novel, network exception handlers leverage end hosts to enforce access policies [24], and some prior work adapts PCA-based anomaly detectors to work well even when data is distributed on many servers [21]. Yet, performing cluster-wide instrumentation of servers to obtain detailed measurements is a novel aspect of this work.

We use the ETW (Event Tracing for Windows [2]) framework to collect socket level events at each server and parse the information locally. Periodically, the measured data is stowed away using the APIs of the underlying distributed file system which we also use for analyzing the data.

In our cluster, the cost of turning on ETW was a median increase of 1.6% in CPU utilization, an increase of 1.2% in disk utilization, 10% more cpu cycles per byte of network traffic and fewer than a 2Mbps drop in network throughput even when the server was using the NIC at capacity (i.e., at 1Gbps). This overhead is low primarily due to the efficient tracing framework [2] underlying ETW but also because unlike packet capture which involves an interrupt from the kernel's network stack for each packet, we use ETW to obtain socket level events, one per application read or write, which aggregates over several packets and skips network chatter. To keep the cumulative data upload rate manageable, we compress the logs prior to uploading. Compression reduces the network bandwidth used by the measurement infrastructure by at least 10x.

In addition to network level events, we collect and use application logs (job queues, process error codes, completion times etc.) to see which applications generate what network traffic as well as how network artifacts (congestion etc.) impact applications.

Over a month, our instrumentation collected nearly a petabyte of uncompressed data. We believe that deep packet inspection is infeasible in production clusters of this scale–it would be hard to justify the associated cost and the spikes in CPU usage associated with packet capture and parsing on the server interfaces are a concern production cluster managers. The socket level detail we collect is both doable and useful, since as we will show next this lets us answer questions that SNMP feeds cannot.

3. APPLICATION WORKLOAD

Before we delve into measurement results, we briefly sketch the nature of the application that is driving traffic on the instrumented cluster. At a high level, the cluster is a set of commodity servers that supports map reduce style jobs as well as a distributed replicated block store layer for persistent storage. Programmers write jobs in a high-level SQL like language called Scope [8]. The scope compiler transforms the job into a workflow (similar to that of Dryad [23]) consisting of phases of different types. Some of the common phase types are *Extract* which looks at the raw data and generates a stream of relevant records, *Partition* which divides a stream into a set number of buckets, *Aggregate* which is the Dryad equivalent of reduce and *Combine* which implements joins. Each phase consists of one or more vertices that run in parallel and perform the same computation on different parts of the input stream. Input data may need to be read off the network if it is not available on the same machine but outputs are always written to the local disk for simplicity. Some phases can function as a pipeline, for example *Partition* may start dividing the data generated by *Extract* into separate hash bins as soon as an extract vertex finishes, while other phases may not be pipelineable, for example, an *Aggregate* phase that computes the median sale price of different textbooks would need to look at every sales record for a textbook before it can compute the median price. Hence, in this case the aggregate can run only after every partition vertex that may output sales records for this book name completes. All the inputs and the eventual outputs of jobs are stored in a reliable replicated block storage mechanism called Cosmos that is implemented on the same commodity servers that do computation. Finally, jobs range over a broad spectrum from short interactive programs that may be written to quickly evaluate a new algorithm to long running, highly optimized, production jobs that build indexes.

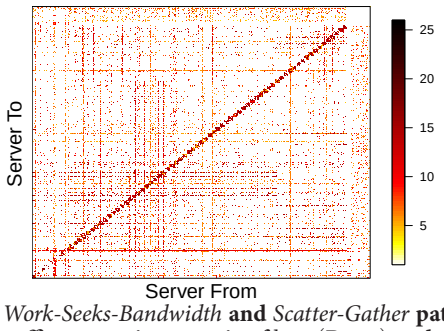

Figure 2: The *Work-Seeks-Bandwidth* and *Scatter-Gather* patterns in datacenter traffic as seen in a matrix of $\log_e$(Bytes) exchanged between server pairs in a representative 10s period. (See §4.1).

4. TRAFFIC CHARACTERISTICS

Context: The datacenter we collect traffic from has the typical structure sketched in Figure 1. Virtualization is not used in this cluster, hence each IP corresponds to a distinct machine which we will refer to as a *server*. A matrix representing how much traffic is exchanged from the server denoted by the row to the server denoted by the column will be referred to as a *traffic matrix* (TM). We compute TMs at multiple time-scales, $1s$, $10s$ and $100s$ and between both servers and top-of-rack (ToR) switches. The latter ToR-to-ToR TM has zero entries on the diagonal, i.e., unlike the server-to-server TM only traffic that flows across racks is included here. By *flow*, we mean the canonical five-tuple (source IP,port, destination IP,port and protocol). When explicit begins and ends of a flow are not available, similar to much prior work [26, 30], we use a long inactivity timeout (default 60s) to determine when a flow ends (or a new one begins). Finally, clocks across the various servers are not synchronized but also not too far skewed to affect the subsequent analysis.

4.1 Patterns

Two pronounced patterns together comprise a large chunk of traffic in the data center. We call these the work-seeks-bandwidth pat-

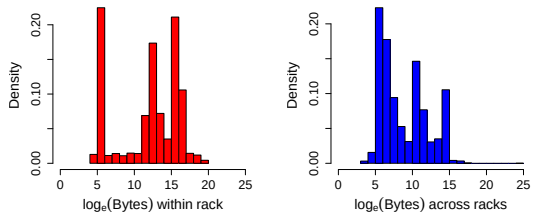

Figure 3: How much traffic is exchanged between server pairs (non-zero entries)?

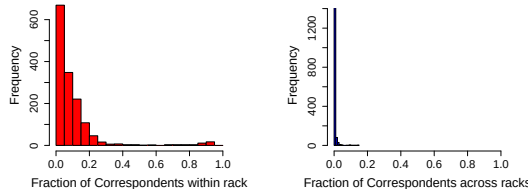

Figure 4: How many other servers does a server correspond with? ($Rack = 20$ servers, $Cluster \sim 1500$ servers)

tern and the scatter-gather pattern due to their respective causes. Figure 2 plots the $\log_e(Bytes)$ exchanged between server pairs in a 10s period. We order the servers such that those within a rack are adjacent to each other on the axes. The small squares around the diagonal represent a large chunk of the traffic and correspond to exchanges among servers within a rack. At first blush, this figure resembles CPU and memory layouts on ASIC chips that are common in the architecture community. Indeed the resemblance extends to the underlying reasons. While chip designers prefer placing components that interact often (e.g., cpu-L1 cache, multiple cpu cores) close by to get high bandwidth interconnections on the cheap, writers of data center applications prefer placing jobs that rely on heavy traffic exchanges with each other in areas where high network bandwidth is available. In topologies such as the one in Figure 1 this translates to the engineering decision of placing jobs within the same server, within servers on the same rack or within servers in the same VLAN and so on with decreasing order of preference and hence the work-seeks-bandwidth pattern. Further, the horizontal and vertical lines represent instances wherein one server pushes (or pulls) data to many servers across the cluster. This is indicative of the map and reduce primitives underlying distributed query processing infrastructures wherein data is partitioned into small chunks, each of which is worked on by different servers, and the resulting answers are later aggregated. Hence, we call this the scatter-gather pattern. Finally, we note that the dense diagonal does not extend all the way to the top right corner. This is because the area on the far right (and far top) corresponds to servers that are external to the cluster which upload new data into the cluster or pull out results from it.

We attempt to characterize these patterns with a bit more precision. Figure 3 plots the log-distribution of the non-zero entries of the TM. At first both distributions appear similar, non-zero entries are somewhat heavy-tailed, ranging from $[e^4 : e^{20}]$ with server pairs that are within the same rack more likely to exchange more bytes. Yet, the true distributions are quite different due to the numbers of zero entries– the probability of exchanging no traffic is 89% for server pairs that belong to the same rack and 99.5% for pairs that are in different racks. Finally, Figure 4 shows the distributions of how many correspondents a server talks with. A server either talks to almost all the other servers within the rack (the bump near 1 in Fig. 4 left) or it talks to fewer than 25% of servers within the rack. Further, a server either doesn't talk to servers outside its rack (the spike at zero in Fig. 4 right) or it talks to about 1-10% of outside servers. The median numbers of correspondents for a server are two (other) servers within its rack and four servers outside the rack.

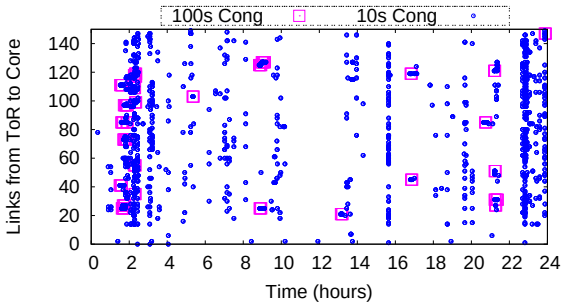

Figure 5: When and where does congestion happen in the datacenter?

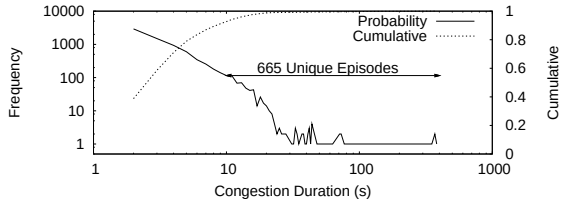

Figure 6: Length of Congestion Events

We believe that figs. 2 to 4 together form the first characterization of datacenter traffic at a macroscopic level and comprise a model that can be used in simulating such traffic.

4.2 Congestion Within the Datacenter

Next, we shift focus to hot-spots in the network, i.e., links that have average utilization above some constant C. Results in this section use a value of $C = 70\%$ but choosing a threshold of 90% or 95% yields qualitatively similar results. Ideally, one would like to drive the network at as high an utilization as possible without adversely affecting throughput. Pronounced periods of low network utilization likely indicate (a) that the application by nature demands more of other resources such as cpu and disk than the network, or (b) that the applications can be re-written to make better use of available network bandwidth.

Figure 5 illustrates when and where links within the monitored network are highly utilized. Highly utilized links happen often! Among the 150 inter-switch links that carry the traffic of the 1500 monitored machines, 86% of the links observe congestion lasting at least 10 seconds and 15% observe congestion lasting at least 100 seconds. Short congestion periods (blue circles, 10s of high utilization) are highly correlated across many tens of links and are due to brief spurts of high demand from the application. Long lasting congestion periods tend to be more localized to a small set of links. Figure 6 shows that most periods of congestion tend to be short-lived. Off all congestion events that are more than one second long, over 90% are no longer than 2 seconds, but long epochs of congestion exist – in one day's worth of data, there were 665 unique episodes of congestion that each lasted more than 10s, a few epochs lasted several hundreds of seconds and the longest lasted for 382 seconds.

When congestion happens, is there collateral damage to victim flows that happen to be using the congested links? Figure 7 compares the rates of flows that overlap high utilization periods with the rates of all flows. From an initial inspection, it appears as if the rates do not change appreciably (see cdf below). Errors such as flow timeouts or failure to start may not be visible in flow rates, hence we correlate high utilization epochs directly with application level logs. Figure 8 shows that jobs experience a median increase of 1.1x in their probability of failing to read input (s) if they have flows traversing high utilization links. Note that while outputs are always written to the local disk, the next phase of the job that uses this data may have to

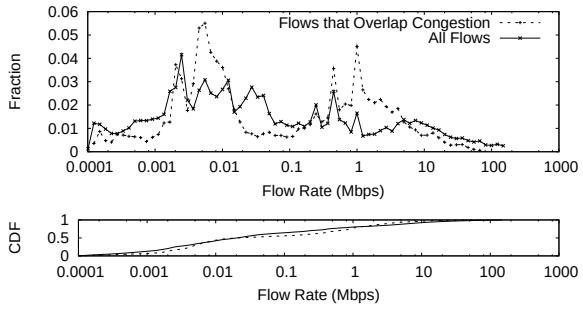

Figure 7: Comparing rates of flows that overlap congestion with rates of all flows.

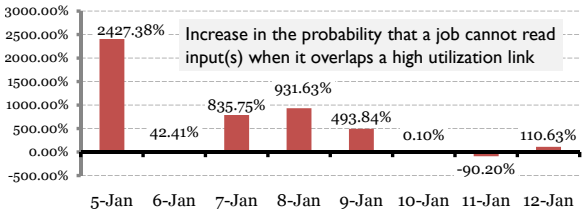

Figure 8: Impact of high utilization– The likelihood that a job fails because it is unable to read requisite data over the network increases by 1.1x (median) during high utilization epochs.

read it over the network if necessary. When a job is unable to find its input data, or is unable to connect to the machine that has the input data, or is stuck, i.e., does not make steady progress in reading more of its input, the job is killed and logged as a *read failure*. We note upfront that not all read failures are due to the network; besides congestion they could be caused by an unresponsive machine, bad software or bad disk sectors. However, we observe a high correlation between network congestion and read failures leading us to believe that a sizable chunk of the observed read failures are due to congestion. Over a one week period, we see that the inability to read input (s) increases when the network is highly utilized. Further, the more prevalent the congestion (on 5'th, 8'th Jan for example), the larger the increase and in particular the days with little increase (10'th, 11'th Jan) correspond to a lightly loaded weekend.

When high utilization epochs happen, we would like to know the causes behind high volumes of traffic. Operators would like to know if these high volumes are normal. Developers can better engineer job placement if they know which applications send how much traffic and network designers can evaluate architecture choices better by knowing what drives the traffic. To attribute network traffic to the applications that generate it, we merge the network event logs with logs at the application-level that describe which job and phase (e.g., map, reduce) were active at that time. Our results show that, as expected, jobs in the *reduce* phase are responsible for a fair amount of the network traffic. Note that in the reduce phase of a map-reduce job, data in each partition that is present at multiple servers in the cluster (e.g., all personnel records that start with 'A') has to be pulled to the server that handles the reduce for the partition (e.g., count the number of records that begin with 'A') [14, 23].

However, unexpectedly, the *extract* phase also contributed a fair amount of the flows on high utilization links. In Dryad [23], extract is an early phase in the workflow that parses the data blocks. Hence, it looks at by far the largest amount of data and the job manager attempts to keep the computation as close to data as possible. It turns out that a small fraction of all extract instances read data off the network if all of the cores on the machine that has the data are busy at the time. Yet another unexpected cause for highly utilized links were *evacuation* events. When a server repeatedly experiences problems, the automated management system in our cluster evacuates all

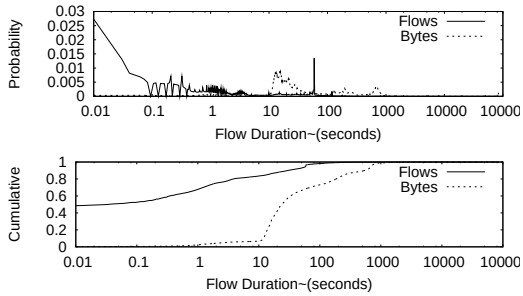

Figure 9: More than 80% of the flows last less than ten seconds, fewer than .1% last longer than 200s and more than 50% of the bytes are in flows lasting less than 25s.

the usable blocks on that server prior to alerting a human that the server is ready to be re-imaged (or reclaimed). The latter two unexpected sources of congestion helped developer's re-engineer the applications based on these measurements.

To sum up, high utilization epochs are common, appear to be caused by application demand and have a moderate negative impact on job performance.

4.3 Flow Characteristics

Figure 9 shows that the traffic mix changes frequently. The figure plots the durations of 165 million flows (a day's worth of flows) in the cluster. Most flows come and go (80% last less than 10s) and there are few long running flows (less than .1% last longer than 200s). This has interesting implications for traffic engineering. Centralized decision making, in terms of deciding which path a certain flow should take, is quite challenging–not only would the central scheduler have to deal with a rather high volume of scheduling decisions but it would also have to make the decisions very quickly to avoid visible lag in flows. One might wonder whether most of the bytes are contained in the long running flows. If this were true, scheduling just the few long running flows would be enough. Unfortunately, this does not turn out to be the case in DC traffic; more than half the bytes are in flows that last no longer than 25s.

Figure 10 shows how the traffic changes over time within the data center. The figure on the top shows the aggregate traffic rate over all server pairs for a ten hour period. Traffic changes quite quickly, some spikes are transient but others last for a while. Interestingly the top of the spikes is more than half the full-duplex bisection bandwidth of the network. Communication patterns that are full duplex are rare, because typically at any time, the producers and consumers of data are fixed. Hence, this means that at several times during a typical day all the used network links run close to capacity.

Another dimension in traffic change is the flux in participants– even when the net traffic rate remains the same, the servers that exchange those bytes may change. Fig 10 (bottom) quantifies the absolute change in traffic matrix from one instant to another normalized by the total traffic. More precisely if $M(t)$ and $M(t+\tau)$ are the traffic matrices at time t and $t+\tau$, we plot

$$Normalized\ Change = \frac{|M(t+\tau) - M(t)|}{|M(t)|},$$

where the numerator is the absolute sum of the entry wide differences of the two matrices and the denominator is the absolute sum of entries in $M(t)$. We plot changes for both $\tau = 100s$ and $\tau = 10s$. At both these time-scales, the median change in traffic is roughly 82% and the 90th and 10th percentiles are 149% and 37% respectively. This means that even when the total traffic in the matrix remains the same (flat regions on the top graph), the server pairs that are involved in these traffic exchanges change appreciably. There

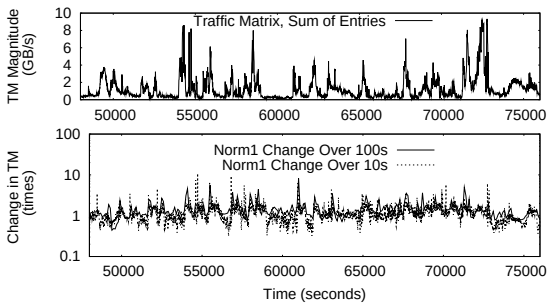

Figure 10: Traffic in the data-center changes in both the magnitude (top) and the participants (bottom).

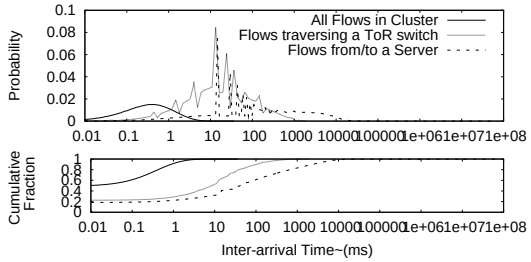

Figure 11: Distribution of inter-arrival times of the flows seen in the entire cluster, at Top-of-Rack switches (averaged) and at servers (averaged).

are instances of both leading and lagging change; short bursts cause spikes at the shorter time-scale (in dashed line) that smooth out at the longer time scale (in solid line) whereas gradual changes appear conversely, smoothed out at shorter time-scales yet pronounced on the longer time-scale. Significant variability appears to be a key aspect of data center traffic.

Figure 11 portrays the distribution of inter-arrival times between flows as seen at hosts in the datacenter. *How long after a flow arrives would one expect another flow to arrive?* If flow arrivals were a Poisson process, network designers could safely design for the average case. Yet, we see evidence of periodic short-term bursts and long tails. The inter-arrivals at both servers and top-of-rack switches have pronounced periodic modes spaced apart by roughly 15 ms. We believe that this is likely due to the stop-and-go behavior of the application that rate-limits the creation of new flows. The tail for these two distributions is quite long as well, servers may see flows spaced apart by up to 10s. Finally, the median arrival rate of all flows in the cluster is 10^5 flows per second, or 100 flows in every millisecond. Centralized schedulers that decide which path to pin a flow on may be hard pressed to keep up. Scheduling application units (jobs etc.) rather than the flows caused by these units is likely to be more feasible, as would distributed schedulers that engineer flows by making simple random choices [25, 36].

4.4 On Incast

We do not see direct evidence of the incast problem [10, 33], perhaps because we don't have detailed TCP level statistics for flows in the datacenter. However, we comment on how often the several assumptions that need to happen in the examined datacenter for incast to occur. First, due to the low round-trip times in datacenters, the bandwidth delay product is small which when divided over the many contending flows on a link results in a small congestion window for each flow. Second, when the interface's queue is full, multiple flows should see their packets dropped. Due to their small congestion windows, these flows cannot recover via TCP fast retransmit, are stuck until a TCP timeout and have poor throughput. Third, for the throughput of the network to also go down, synchronization should happen such that no other flow is able to pick up the slack when some flows are in TCP timeout. Finally, an application is impacted more if it cannot make forward progress until all its network flows finish.

MapReduce, or at least the implementation in our datacenter, exhibits very few scenarios wherein a job phase cannot make incremental progress with the data it receives from the network. Further, two engineering decisions explicitly limit the number of mutually contending flows–first, applications limit their simultaneously open connections to a small number (default to 4) and second, computation is placed such that with a high probability network exchanges are local (i.e., within a rack, within a VLAN etc. Figure 2). This local nature of flows, most are either within the same rack or VLAN, implicitly isolates flows from other flows elsewhere in the network and reduces the likelihood that a bottleneck-ed switch will carry the large number of flows needed to trigger incast. Finally, several jobs run on the cluster at any time. Though one or a few flows may suffer timeouts, this multiplexing allows other flows to use up the bandwidth that becomes free thereby reducing the likelihood of wholesale throughput collapse.

We do believe that TCP's inability to recover from even a few packet drops without resorting to timeouts in low bandwidth delay product settings is a fundamental problem that needs to be solved. However on the observed practical workloads, which is perhaps typical of a wide set of datacenter workloads, we see little evidence of throughput collapse due to this weakness in TCP.

5. TOMOGRAPHY IN THE DATA CENTER

Socket level instrumentation, which we used to drive the results presented so far in the paper, is unavailable in most datacenters but link counters at routers (e.g., SNMP byte counts) are widely available. It is natural to ask – in the absence of more detailed instrumentation, to what approximation can we achieve similar value from link counters? In this section, we primarily focus on network tomography methods that infer traffic matrices (origin-destination flow volumes) from link level SNMP measurements [20, 35]. If these techniques are as applicable in datacenters as they are in ISP networks, they would help us unravel the nature of traffic in many more datacenters without the overhead of detailed measurement.

There are several challenges for tomography methods to extend to data centers. Tomography inherently is an *under-constrained problem*; while the number of origin-destination flow volumes to be estimated is quadratic ($n(n-1)$), the number of link measurements available (i.e., constraints) is much fewer, often a small constant times the number of nodes. Further, the typical datacenter topology (Fig. 1) represents a worst-case scenario for tomography. As many ToR switches connect to one or a few high-degree aggregation switches, the number link measurements available is small (typically $2n$). To combat this under-constrained nature, tomography methods model the traffic seen in practice and use these models as *apriori* estimates of the traffic matrix, thereby narrowing the space of TMs that are possible given the link data.

A second difficulty stems from the fact that many of the priors that are known to be effective make simplifying assumptions. For example, the gravity model assumes that the amount of traffic a node (origin) would send to another node (destination) is proportional to the traffic volume received by the destination. Though this prior has been shown to be a good predictor in ISP networks [20, 35], the pronounced patterns in traffic that we observe are quite far from the simple spread that the gravity prior would generate. A final difficulty is due to scale. While most existing methods can compute traffic matrices between a few 100 participants (e.g., POPs in an ISP), even a reasonable cluster has several thousand servers.

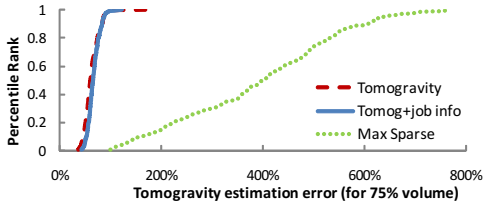

Figure 12: CDF of estimation error for TMs estimated by (i) tomogravity, (ii) tomogravity augmented with job information, and (iii) sparsity maximization.

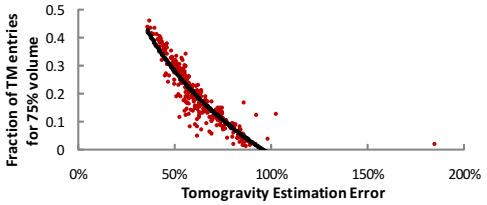

Figure 13: The fraction of entries that comprise 75% of the traffic in the ground truth TM correlates well (negatively) with the estimation error of tomogravity.

Methodology: We compute link counts from the ground truth TM and measure how well the TM estimated by tomography from these link counts approximates the true TM. Our error function avoids penalizing mis-estimates of matrix entries that have small values [35]. Specifically, we choose a threshold T such that entries larger than T make up about 75% of traffic volume and then obtain the Root Mean Square Relative Error (RMSRE) as $\sqrt{\sum_{x_{ij}^{true} \geq T} \left(\frac{x_{ij}^{est} - x_{ij}^{true}}{x_{ij}^{true}} \right)^2}$, where $x_{ij}^{true}, x_{ij}^{est}$ are the true and estimated entries respectively. This evaluation sidesteps the issue of scale by attempting to obtain traffic matrices at the ToR level. We report aggregate results over 288 *ToR-level* TMs, which is about a day's worth of 5 min average TMs.

5.1 Tomogravity

Tomogravity based tomography methods [35] use the gravity traffic model to estimate apriori the traffic between a pair of nodes. In Figure 12, we plot the CDF of tomogravity estimation errors of 5 min TMs taken over an entire day. Tomogravity results in fairly inaccurate inferences, with estimation errors ranging from 35% to 184% and a median of 60%. We observed that the gravity prior used in estimation tends to spread traffic around whereas the ground truth TMs are sparse. An explanation for this is that communication is more likely between nodes that are assigned to the same job rather than all nodes, whereas gravity model, not being aware of these job-clusters, introduces traffic across clusters, thus resulting in many non-zero TM entries. To verify this conjecture, we show, in Figure 13, that the estimation error of tomogravity is correlated with the sparsity of the ground truth TM – the fewer the number of entries in ground truth TM the larger the estimation error. (A logarithmic best-fit curve is shown in black.)

5.2 Sparsity Maximization

Given the sparse nature of datacenter TMs, we consider an estimation method that favors sparser TMs among the many possible. Specifically, we formulated a mixed integer linear program (MILP) that generates the sparsest TM subject to link traffic constraints. Sparsity maximization has been used earlier to isolate anomalous traffic [34]. However, we find that the sparsest TMs are much sparser than ground truth TMs (see Figure 14) and hence yield a worse estimate than tomogravity (see Figure 12). The TMs estimated via sparsity maximization contain typically 150 non-zero entries, which is about 3% of the total TM entries. Further, these non-zero entries do not correspond to heavy hitters in the ground truth TMs–only a handful (5-20) of these entries correspond to entries in ground truth TM with value greater than the 97-th percentile. Sparsity maximization appears overly aggressive and datacenter traffic appears to be somewhere in between the dense nature of tomogravity estimated TMs and the sparse nature of sparsity maximized TMs.

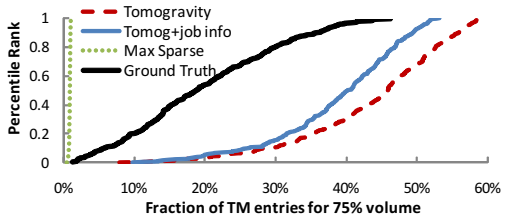

Figure 14: Comparing the TMs estimated by various tomography methods with the ground truth in terms of the number of TM entries that account for 75% of the total traffic. Ground truth TMs are sparser than tomogravity estimated TMs, and denser than sparsity maximized estimated TMs.

5.3 Prior based on application metadata

Can we leverage application logs to supplement the shortcomings of tomogravity? Specifically, we use metadata on which jobs ran when and which machines were running instances of the same job. We extend the gravity model to include an additional multiplier for traffic between two given nodes (ToRs) i and j that is larger if the nodes share more jobs and fewer otherwise, i.e., the product of the number of instances of a job running on servers under ToRs i and j, summed over all jobs k. In practice however, this extension seems to not improve vanilla tomogravity by much, the estimation errors are only marginally better (Figure 12) though the TMs estimated by this method are closer to ground truth in terms of sparsity (Figure 14). We believe that this is due to nodes in a job assuming different roles over time and traffic patterns varying with respective roles. As future work, we plan to incorporate further information on roles of nodes assigned to a job.

6. RELATED WORK

Data center networking has recently emerged as a topic of interest. There is not much work on measurement, analysis, and characterization of datacenter traffic. Greenberg *et al.* [18] report datacenter traffic characteristics–variability at small timescales and statistics on flow sizes and concurrent flows, and use these to guide network design. Benson *et al.* [7] perform a complementary study of traffic at the edges of a datacenter by examining SNMP traces from routers and identify ON-OFF characteristics whereas this paper examines novel aspects of traffic within a data center in detail.

Traffic measurement in enterprises is better studied with papers that compare enterprise traffic to wide-area traffic [28], study the health of an enterprise network based on the fraction of successful flows generated by end-hosts [19] and use traffic measurement on end-hosts for fine-grained access control [24].

7. DISCUSSION

We believe that our results here would extend to other *mining* data centers that employ some flavor of map-reduce style workflow computation on top of a distributed block store. For example, several companies including Yahoo! and Facebook have clusters running Hadoop, an open source implementation of map-reduce and Google has clusters that run map reduce. In contrast, *web* or *cloud*

data centers that primarily deal with generating responses for web requests (e.g., mail, messenger), are likely to have different characteristics. Our results are primarily dictated by how the applications have been engineered and are likely to hold even as the specifics such as network topology and over-subscription ratios change. However, we note that pending future data center measurements, based perhaps on instrumentation similar to that described here, these beliefs remain conjectures at this point.

An implication of our measurements is worth calling out. By partitioning the measurement problem which in the past was done at switches or routers across many commodity servers we relax many of the typical constraints (memory, cycles) for measurement. Clever counters or data structures to perform measurement at line speed under constrained memory are no longer as crucial but continue to be useful in keeping overheads small. Conversely, however, handling scenarios where multiple independent parties are each measuring a small piece of the puzzle gains new weight.

8. CONCLUSIONS

In spite of widespread interest in datacenter networks, little has been published that reveals the nature of their traffic, or the problems that arise in practice. This paper is a first attempt to capture both the macroscopic patterns – which servers talk to which others, when and for what reasons – as well as the microscopic characteristics – flow durations, inter-arrival times, and like statistics – that should provide a useful guide for datacenter network designers. These statistics appear more regular and better behaved than counterparts from ISP networks (e.g., "elephant" flows only last about one second). This, we believe, is the natural outcome of the tighter coupling between network, computing, and storage in datacenter applications. We did not see evidence of super large flows (flow sizes being determined largely by chunking considerations, optimizing for storage latencies), TCP incast problems (the preconditions apparently not arising consistently), or sustained overloads (owing to near ubiquitous use of TCP). However, episodes of congestion and negative application impact do occur, highlighting the significant promise for improvement through better understanding of traffic and mechanisms that steer demand.

Acknowledgments: We are grateful to Igor Belianski and Aparna Rajaraman for invaluable help in deploying the measurement infrastructure and also to Bikas Saha, Jay Finger, Mosha Pasumansky and Jingren Zhou for helping us interpret results.

9. REFERENCES

[1] Amazon Web Services. http://aws.amazon.com.
[2] EventTracing for Windows. http://msdn.microsoft.com/en-us/library/ms751538.aspx.
[3] Google app engine. http://code.google.com/appengine/.
[4] Hadoop distributed filesystem. http://hadoop.apache.org.
[5] Windows Azure. http://www.microsoft.com/azure/.
[6] L. A. Barroso and U. Hölzle. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. *Synthesis Lectures on Computer Architecture*, 2009.
[7] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding Datacenter Traffic Characteristics. In *SIGCOMM WREN workshop*, 2009.
[8] R. Chaiken, B. Jenkins, P. Åke Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets. In *VLDB*, 2008.
[9] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In *OSDI*, 2006.
[10] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *SIGCOMM WREN Workshop*, 2009.
[11] Cisco Guard DDoS Mitigation Appliance. http://www.cisco.com/en/US/products/ps5888/.
[12] Cisco Nexus 7000 Series Switches. http://www.cisco.com/en/US/products/ps9402/.
[13] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *SIGMOD*, 2003.
[14] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.
[15] N. Duffield, C. Lund, and M. Thorup. Estimating Flow Distributions from Sampled Flow Statistics. In *SIGCOMM*, 2003.
[16] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a Better NetFlow. In *SIGCOMM*, 2004.
[17] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *SOSP*, 2003.
[18] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *ACM SIGCOMM*, 2009.
[19] S. Guha, J. Chandrashekar, N. Taft, and K. Papagiannaki. How Healthy are Today's Enterprise Networks? In *IMC*, 2008.
[20] A. Gunnar, M. Johansson, and T. Telkampi. Traffic Matrix Estimation on a Large IP Backbone - A Comparison on Real Data. In *IMC*, 2004.
[21] L. Huang, X. Nguyen, M. Garofalakis, J. Hellerstein, M. Jordan, M. Joseph, and N. Taft. Communication-Efficient Online Detection of Network-Wide Anomalies. In *INFOCOM*, 2007.
[22] IETF Working Group IP Flow Information Export (ipfix). http://www.ietf.org/html.charters/ipfix-charter.html.
[23] M. Isard, M. Budiu, Y. Yu, A. Birrell, , and D. Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *EUROSYS*, 2007.
[24] T. Karagiannis, R. Mortier, and A. Rowstron. Network exception handlers: Host-Network Control in Enterprise Networks. In *SIGCOMM*, 2008.
[25] M. Kodialam, T. V. Lakshman, and S. Sengupta. Efficient and Robust Routing of Highly Variable Traffic. In *HotNets*, 2004.
[26] R. Kompella and C. Estan. Power of Slicing in Internet Flow Measurement. In *IMC*, 2005.
[27] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A Not-So-Foreign Language for Data Processing. In *SIGMOD*, 2008.
[28] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A First Look at Modern Enterprise Traffic. In *IMC*, 2005.
[29] IETF Packet Sampling (Active WG). http://tools.ietf.org/wg/psamp/.
[30] S. Kandula and D. Katabi and S. Sinha and A. Berger. Dynamic Load Balancing Without Packet Reordering. In *CCR*, 2006.
[31] sFlow.org. Making the network visible. http://www.sflow.org.
[32] A. Soule, A. Lakhina, N. Taft, K. Papagiannaki, K. Salamatian, A. Nucci, M. Crovella, and C. Diot. Traffic Matrices: Balancing Measurements, Inference and Modeling. In *ACM SIGMETRICS*, 2005.
[33] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Andersen, G. Ganger, G. Gibson, and B. Mueller. Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication. In *SIGCOMM*, 2009.
[34] Y. Zhang, Z. Ge, A. Greenberg, and M. Roughan. Network Anomography. In *IMC*, 2005.
[35] Y. Zhang, M. Roughan, N. C. Duffield, and A. Greenberg. Fast accurate computation of large-scale IP traffic matrices from link loads. In *ACM SIGMETRICS*, 2003.
[36] R. Zhang-Shen and N. McKeown. Designing a Predictable Internet Backbone Network. In *HotNets*, 2004.