Mission Control: Enabling Efficient and Heterogeneous Data Transfers in Data Intensive Computing

Zubair Nabi
Robinson College

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science (Option B)

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: Zubair.Nabi@cl.cam.ac.uk

June 14, 2012

Declaration

I Zubair Nabi of Robinson College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 0

Signed:

Date:

Acknowledgements

First and foremost, I would like to thank my supervisors, Dr Steven Hand and Dr Anil Madhavapeddy, for their guidance and stimulation throughout the course of this work. Dr Hand was instrumental in transforming a rough idea into a full-fledged framework. His emphasis on a sound evaluation motivated me to look at every system artifact from the point of view of an eventual experiment. Dr Madhavapeddy was helpful in setting up the experimental environment and providing insight into operating system, virtualization and network stack internals.

This work would be nowhere without the insight provided into CIEL internals by Malte Schwarzkopf, Derek Murray, and Chris Smowton. In addition, Dr Kenneth Birman (Cornell) deserves a mention for his help and patience while I wrestled with Isis2 and Mono.

Furthermore, friends at Cambridge who have metamorphosed my sojourn here into a tour de force – socially, academically, and intellectually – merit my special gratitude. Especially, Natalie Dobson for being a constant source of inspiration and energy. My coursemates in the MPhil Advanced Computer Science have also never ceased to amaze me with their aptitude and support. In the same vein, everyone in the Computer Laboratory – from the amazing teachers and researchers to the extremely invaluable staff – deserves a thank you for their encouragement and help.

Last, but by no means least, I would like to thank my family who form the rock on which I balance my existence. This dissertation is dedicated to them.

Abstract

This is the abstract. Write a summary of the whole thing. Make sure it fits in one page.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the course of the last decade, proliferation in web applications (Facebook, Twitter etc.), and scientific computing (sensor networks, high energy physics, etc.) has resulted in an unparalleled surge in data on the exabyte scale. Consequently, the need to store and analyze this data has engendered an ecosystem of distributed file systems [1], structured and unstructured data stores [2, 3], and data-intensive computing frameworks [4, 5, 6]. These "shared-nothing" frameworks are mostly run in massive data centers with tens of thousands of servers and network elements. These data centers are globally distributed, for diversity and redundancy, and run a wide-variety of loads; including user-facing applications, such as email; custom enterprise applications, such as authentication services; and large-scale batch-processing applications, such as web indexing [7]. The goal of each data center is to provide a multi-tenant[1] environment for energy-, power-, space-, and cost-efficient computation, and storage at scale [8].

Typically, data centers are built using commodity, off-the-shelf servers and switches to find a sweet spot between performance and cost, and to leverage economies of scale [9]. Servers in this model store or process a *chunk*

[1]Multi-tenancy applies to both public clouds, such as Amazon EC2 and Microsoft Azure, where the resources are shared by paying customers and to private clouds, maintained by Google, Facebook, etc., where different departments share the same infrastructure.

of data, with redundancy ensured via replication without any cluster/data center wide memory or storage area network (SAN). Due to their commodity nature, servers and network elements are failure-prone and a failure in say, a *core switch* can bring down the entire network. Therefore, storage and processing frameworks use a range of methods to deal with this, including replication, re-execution, and speculative execution. On the network side, end-hosts running TCP/IP are connected through cheap switches and wired links, although wireless [10] and optical [11, 12] links have recently been explored. Most data centers are provisioned with 1Gbps and 10Gbps switches with L2 Ethernet switched fabric interconnects as opposed to fabrics from the high performance computing community such as InfiniBand. Likewise, due to the prohibitive cost of high-end switches and routers, as opposed to commodity off-the-shelf hardware, bandwidth is a constrained resource in the data center. To ensure optimum utilization, subject to organizational goals, oversubscription factors[2] in real data centers vary significantly: Some are as low as 2:1 while some can get as high as 147:1 [7]. In the cluster hierarchy, bandwidth increases from the edge of the network towards the core (off-rack oversubscription). As a result of off-rack oversubscription, applications are designed to keep cross-rack communication to a minimum [4].

Concurrently, the emergence of cloud computing has resulted in multi-tenant data centers in which user applications have wide-varying requirements and communication patterns. Applications range from low-latency query jobs to bandwidth-hungry batch processing jobs [14]. In addition, applications exhibit one-to-many, many-to-one, and many-to-many communication patterns. At the data center fabric level, to ensure better fault-tolerance, scalability, and end-to-end bandwidth, several new topologies [13, 15, 16, 17] have been proposed to replace the status quo: a hierarchical 2/3-level tree which connects end-hosts through *top-of-rack* (or *aggregation*), and *core* switches. In the same vein, L2 [18, 19] and L3/L4 [14, 20, 21, 22, 23] protocols, topologies and design frameworks [24, 13, 15, 16, 17, 25, 26], and control and vir-

[2]Oversubscription factor is defined as "the ratio of worst-case achievable aggregate bandwidth among the end hosts to the total bisection bandwidth of a particular communication topology" [13].

tualization planes [27, 28, 29, 30, 31, 32, 33] for data center communication and resource allocation have also experienced innovation, focusing on some combination of high bandwidth, fairness, fault-tolerance, and scalability.

In this thesis, we explore a new point in the design space by enabling diversity at the transport layer. Specifically, we propose a decoupling between data transfer policy and mechanism in data intensive computing. To this end, we present the design, implementation, and evaluation of *Mission Control*, an abstraction framework that resides at the application layer and selects the transport semantics for a particular transfer. Based on runtime parameters, underlying topology, and user-supplied constraints, it selects both the transport protocol and the schedule for a particular data transfer. Data intensive computing frameworks such as MapReduce [4] and CIEL [6] delegate data transfers to Mission Control by specifying a set of files and network destinations. In addition, frameworks can also expose a set of constraints, such as the use of encryption. Mission Control uses global information to choose an optimum policy for each data transfer. We have implemented Mission Control for the CIEL [6] framework. Finally, we evaluate the performance of CIEL under various transport semantics.

1.1 The Case Against TCP in the Data Center

During the last 3 decades the Internet protocol stack has experienced considerable innovation. To keep up with the diverse range of ever-emerging applications, a number of physical, data link, and application layer protocols have been developed and have stood the test of time. In contrast, IP, TCP, and UDP have been entrenched at the network and transport layer, leading to an hourglass shaped stack [34]. This ossification of the Internet protocol stack at the transport and network layer has also been mimicked by data center networks. TCP/IP has become the de-facto "narrow-waist" of the data center network. On the one hand, this has standardized direct

communication to/from and within the data center without the need for any middleware, but on the other hand, it has also spawned a new set of problems unique to the data center environment. In the data center, the role of IP at the network layer is minimal, as it is only employed for naming and addressing. The only obstacle is address assignment which needs to be done manually to encode location/topology information within the address[3]. Although manual assignment is tedious and error-prone, it can be automated by leveraging the well-defined structure of data centers [26]. In contrast, TCP faces a number of pathological challenges due to a significantly different bandwidth-delay product, round-trip time (RTT), and retransmission timeout (RTO) [35] than would be experienced in a wide area network (WAN). For example, due to the low RTT, the congestion window for each flow is very small. As a result, flow recovery through TCP fast retransmit is impossible, leading to poor net throughput [36]. To exacerbate problems, data-intensive computing frameworks (DICF) are bandwidth hungry and require all data to be *materialized* before the next stage can commence (For example, *shuffle* between Map and Reduce stages). In fact, data transfer between stages can account for more than 50% of the job completion time [37]. Figure 1.1 shows the CDF of a MapReduce sort application for different sized datasets (between 2 GB and 20 GB). From the graph, it is visible that shuffle can account for up to 60% of total execution time for an I/O intensive application such as sort. Similar to the problem of compute, memory, and disk I/O "stragglers" [38], an imbalance in network efficiency can lead to the job completion time being dictated by the task with the longest shuffle time.

It is important to highlight that some of these problems, stem at least in part from to the difference in traffic characteristics and scale between data centers and other networks. In addition, unlike wide area network applications, there is a tight coupling between an application's usage of network, compute, and storage resources. In production data centers, due to the widely-varying

[3]DHCP can be used for address assignment if only physical IDs, such as MAC addresses, are used throughout. Advanced topologies, such as BCube, DCell, and VL2, operate on logical IDs, which requires an additional physical-to-logical address encoding at the DHCP server.

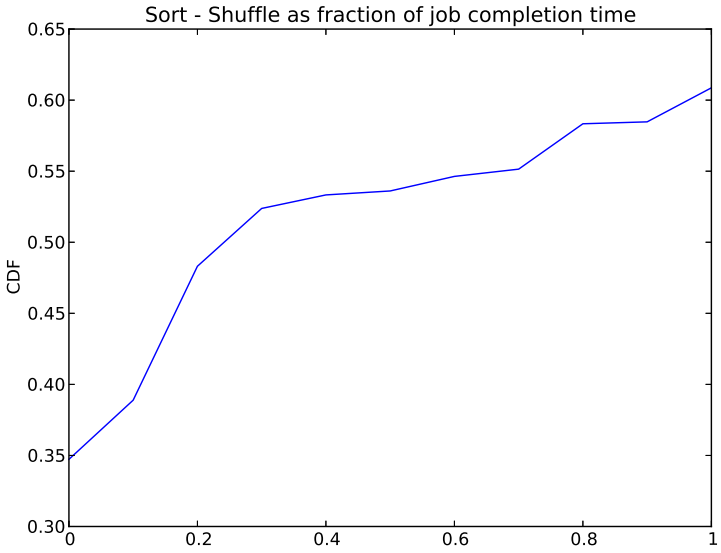


Figure 1.1: Shuffle time as fraction of execution time for Sort

mix of applications, congestion in the network can last from 10s to 100s of seconds [36]. Moreover, during high utilization periods, the task failure rate of batch-processing systems increases, especially during the *reduce* phase of MapReduce-style applications, by a median value of 1.1x, due to read failures over the network. In the worst case, this can halt the entire job if subsequent tasks are dependent on the failed ones. Furthermore, link utilization is highest in the core of the network, but most losses occur towards the edge [7]. Additionally, in some cases over-provisioning the network is overkill and good performance can be achieved by adding a few extra links at hotspot points [39]. Finally, in commodity switches the buffer pool is shared by all interfaces. As a result, if long flows hog the memory, queues can build up for the short flows. In reaction, TCP reduces the window size by half. This leads to a large mismatch between the input rate and the capacity of the link, leading to buffer underflows and loss of throughput [14]. Overall, traffic engineering in the data center is a complex task due to the wide variability in flow duration, flow size, flow arrival time, and server participation [36]: Standard TCP without any global knowledge is suboptimal [40].

Another major shortcoming of TCP, which is manifest in the data center, is TCP throughput collapse, or *incast* [35, 20, 23, 14]. This can cause overall application throughput to decrease by up to 90% [20]. Typically, to handle incast, application-level remedies are added which include varying the size of the packets so that a large number of packets can fit the switch memory, and adding application-level jitter to decorrelate the packet inter-arrival time. Both of these solutions are suboptimal, as they either increase the number of packets, or add unnecessary delay [14]. Finally, most applications operate in a virtualized environment where a single host runs tens of virtual machines (VM). This sharing of CPU by multiple VMs increases the latency experienced by each VM, which can be orders of magnitude higher than the RTT between hosts inside a data center [41, 42]. This significantly slows down the progress of TCP connections between different applications. So much so that there are periods where TCP throughput can go down to zero [43].

All of these problems have even prompted large-scale deployments to abandon TCP altogether. For instance, Facebook [44] now uses a custom UDP transport. Some researchers have gone to the extent of suggesting that fixing TCP is tantamount to "saving the world" [45][4]. For completeness, it is worth noting that TCP was designed as a general transport mechanism for a wide area network (WAN) [46] and has experienced near universal adoption in environments as diverse as satellites [47]. This is primarily due to TCP's ability to provide reliability, congestion and flow control, and in-order packet delivery. Moreover, its maturity makes it a "kitchen sink" solution for developers [20]. Naturally, it has become the transport of choice for the data center. But it is clear from the discussion above that TCP is sub-optimal in a data center environment – a fact agreed upon universally by researchers. As a result, in recent years, a number of novel versions of TCP have been proposed [14, 23, 22, 20, 35]. Unfortunately, most solutions are tied either to the underlying topology or the structure and traffic mix of the data center: There is no one-size-fits-all solution.

[4]Heat and CO_2 emissions and radioactive waste can be accounted to TCP's inefficiency.

1.2 Augmenting the Narrow-waist of the Data Center

It should be clear from the discussion above that the data center communication optimization space is wide due to the interplay of data center structure, traffic patterns, and application requirements. Traffic engineering that performs optimal routing in the data center requires global state, multipath routing, and the assumption of short-term predictability for adaptation [40, 48]. Further, data intensive computing frameworks (DICF) work at the level of a *transfer*, which is not covered by existing solutions which operate at the flow or packet level [37]. Put differently, stages between DICF are dependent on bulk *transfers* of data. These requirements suggest that there needs to be a decoupling between data transfer policy and mechanism. Therefore, in light of these requirements, this thesis presents the design, implementation, and evaluation of *Mission Control*, an abstraction layer which operates between the application layer and network layer to choose the optimum transport protocol. Specifically, it uses runtime parameters, user-supplied requirements, network layout, and application semantics to dynamically choose the underlying transport protocol and schedule data transfer. The same strategy when applied at the network layer improves both application-level and network-level performance [49]. In addition to improving performance, transport layer characteristics can also be chosen to improve energy efficiency [50] or security [51]. Adding power constraints can greatly reduce the consumption footprint of today's data centers where the network consumes 10-20% of the total power [52]. Likewise, application traffic can transparently and efficiently [51] be encrypted to ensure security in a multi-tenant environment.

Our evaluation shows that.....

The rest of this document is structured as follows. §2 gives an overview of the cloud computing paradigm, data intensive computing frameworks, and application models in the cloud. Related work is summarized in §3. §4 presents the design and implementation of Mission Control. An evaluation

of the network patterns of existing systems and of Mission Control is given in §5. Finally, §6 concludes and points to future work.

Chapter 2

Background

This section first gives an overview of cloud computing (§2.1), followed by an introduction to data intensive computing frameworks (§2.2), and finally, a discussion on consistency, replication, scalability, and reliability in the context of cloud computing (§2.3).

2.1 Cloud Computing

Cloud computing is an all encompassing term for applications (services from the point of view of the average user), software systems, and the hardware in massive data centers [53]. *Public clouds* are used by companies to provide utility computing in which software (SaaS), infrastructure (IaaS), and platform (PaaS) can be a service. Similarly, organizations and companies also maintain *Private clouds* which are used internally for data processing and storage. In the cloud computing model, users can buy X as a service (XaaS), where X can be software, infrastructure, and platform and elastically increase or decrease required resources based on dynamic requirements. This allows the user to focus on application design without having to worry about system or network administration. The rise of cloud computing has been fuelled by the rise of "Big Data" and in turn, data intensive computing. The

Cloud has become the ideal platform to store and analyze massive datasets generated by applications as diverse as satellite imagery and web indexing. Batch processing systems, such as MapReduce [4], are in turn used to compute user queries over these datasets. Finally, users can leverage the "cost associativity"[1] of the Cloud and the parallelism in the application to speed up the job completion time. Data intensive computing frameworks are the subject of the next section.

2.2 Data Intensive Computing

Google's MapReduce [4] and its open source counterpart, Hadoop [54], are at the forefront of data intensive computing. MapReduce is a programming model and execution engine for performing large-scale computation on a cluster of commodity machines. It abstracts work division, communication, and fault tolerance beneath a two function API. The user only needs to specify one *map* and one *reduce* function, while the framework takes care of the rest. The architecture consists of a master node and a set of worker nodes. The master node schedules shards of the input dataset – stored on a distributed filesystem – on the worker nodes. The entire execution is two-phased: first the map function is applied to all chunks of input data by the worker nodes, followed by the application of the reduce function to the output of the mappers. The framework operates on key-value pairs. The intermediate key-value pairs, obtained after the map phase, are *shuffled* to a set of reduce workers based on hash partitioning. The framework also has several optimizations such as: 1) Locality: input data is assigned on the basis of locality, 2) Fault-tolerance: failed tasks are simply rescheduled, and 3) Speculative re-execution: slow executing tasks are speculatively re-executed on other nodes to improve the job completion time.

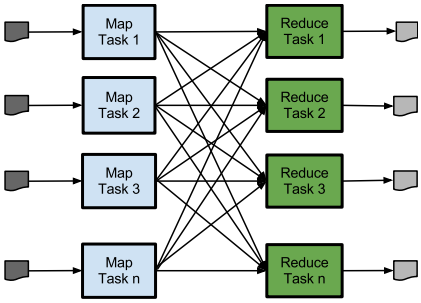
One major shortcoming of MapReduce is that it is strictly a two-phase architecture, as a result of which, it cannot easily express multi-phase SQL-like

[1] Using a large number of computers for a short time costs the same amount as using a small number for a long time.

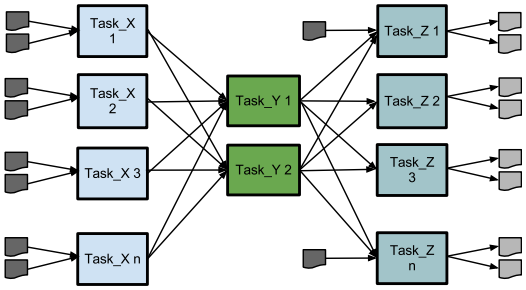
queries. Implementing these queries is awkward in the MapReduce paradigm as they constitute multiple MapReduce jobs, requiring external operators to merge the output [55]. Additionally, the framework is also handicapped by its single input, single output restriction. These shortcomings are addressed by Microsoft's Dryad system [5]. Dryad is a general purpose framework that supports the processing of algorithms with directed cyclic graph (DAG) flows. Like MapReduce, the framework abstracts work scheduling, data partitioning, fault tolerance, and communication beneath a data-flow execution engine. To this end, the developer only needs to (a) construct a DAG of processing using provided primitives, and (b) provide a binary executable for nodes in the DAG. Each vertex in the graph represents the execution of some operation over its input data, while the edges between the vertices represent the communication patterns. The framework automatically schedules each vertex to a machine/core and sets up the communication channels. It is important to highlight that unlike MapReduce, Dryad vertices can have multiple inputs and outputs depending on the programming semantics. Finally, developers can easily use high-level languages to restrict the programming interface for a particular domain.

While MapReduce and Dryad are efficient at processing a rich set of applications with sequential flow, they fall short of supporting iterative and recursive applications [56, 57]. In addition, these applications, such as k-means clustering, PageRank, etc., require a more expressive language to define their flow. These problems are addressed by CIEL [6] which includes two components: (1) A scripting language, *Skywriting* [58], and (2) A distributed execution engine. Skywriting is a Turing-complete language that can be used to express data-dependent control flow. It provides primitives to loop over expressions or call them recursively. In addition, scripting primitives can be used to spawn and execute tasks, and de-reference data (reminiscent of pointers in C/C++). Further, it provides library functions to express the flow of other frameworks such as MapReduce, as CIEL subsumes both MapReduce and Dryad. Architecturally, CIEL is similar to MapReduce and Dryad, in that it also has a single master and several worker nodes, with the former in charge

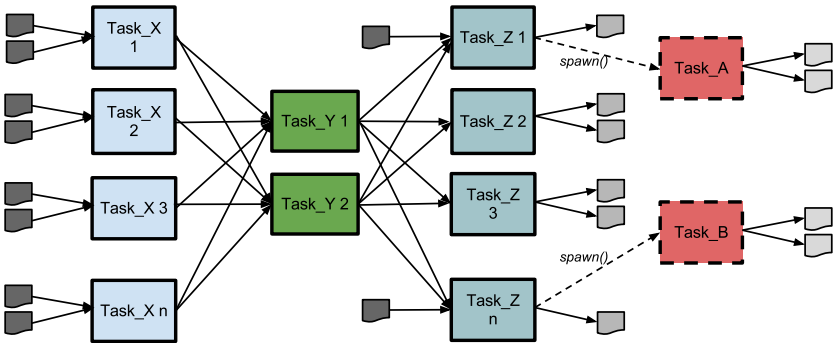
of computation state, and the latter in charge of the actual computation. CIEL relies on three primitives – objects, references, and tasks – for dynamic task graphs. Specifically, objects are data sequences that are used for input and are generated as output. Objects which have not been fully computed yet, can be used as references. Further, tasks are the unit of computation and scheduling. Tasks use programmatic code, or binaries to execute over all its dependencies. Moreover, tasks can either compute all its output objects, or alternatively, spawn other tasks to do so. Tasks can be executed in either *eager*, or *lazy* fashion, with the latter being the default strategy. Finally, CIEL uses a number of optimizations to improve performance such as deterministic naming, and streaming. Figure 2.1 highlights the major differences between the three frameworks.



(a) MapReduce has two stages with single input and single output.



(b) Dryad execution can consist of multiple stages with multiple input and multiple output.



(c) CIEL execution, which subsumes MapReduce and Dryad, can consist of multiple stages with multiple input and multiple output and tasks can dynamically spawn other tasks as well.

Figure 2.1: Overview of data intensive computing frameworks.

There are also high-level languages [59, 60, 58, 61] which allow users to write applications which are transparently translated into a data-flow plan, and then scheduled on top of the underlying framework. In addition, there is a rich body of work which aims to reduce the turn-around time of these systems [62, 63, 38]. While these proposals have been successful in improving scheduling and fairness, the network utilization of these batch-processing systems has largely been ignored.

As mentioned earlier, these frameworks operate over large amounts of data inside environments which are error-prone. Therefore, they have to make trade-offs between consistency, replication, scalability, and reliability. These issues are discussed in the next section.

2.3 Consistency, Replication, Scalability, and Reliability in the Cloud

It is a well-known principle that it is hard to build distributed systems which are consistent, available, reliable, and scalable. In fact, according to Brewer's CAP theorem, it is impossible to build a distributed system which has consistency (all nodes see the same data), availability (the service is always accessible), and partition tolerance (the services can tolerate the loss of nodes/data) at the same time [64, 65]. Developers have to sacrifice one of the three. Due to the failure-prone, commodity infrastructure used to build distributed systems today, partition tolerance is a given. Therefore, systems have to make a choice between consistency and availability [66]. Most cloud applications today, such as Amazon's Dynamo [3] focus on availability resulting in weak consistency or *eventual consistency*. This basically available, soft state, eventually consistent (BASE) [67] model enables applications to function even in the face of partial failure. To this end, applications operate on local (possibly stale) replicas, while the underlying system asynchronously applies updates to replicas.

One way to implement replication updates is through *virtual synchrony* [68]. In this model, processes are organized into dynamic *process groups*, and messages are exchanged between groups, not individual processes, with ordering guarantees. From the perspective of the application, all changes to the shared data appear to be synchronous. As process groups can have names, they can be treated as topics in a publish-subscribe system [69]. In addition, processes within the group are informed if group membership changes to ensure a consistent view of group membership across all members. Whenever a new member joins a process group, the current state is checkpointed and sent to the newly joined member to initialize its group replica in one atomic operation. Processes can also send multicast events to groups with ordering guarantees.

These issues are important for a distributed system such as Mission Control, as performance can be dependent on the durability and consistency of control traffic. As mentioned earlier, the traffic matrix in a data center can change in a matter of seconds, therefore, Mission Control needs to be able to respond in a timely fashion.

Chapter 3

Related Work

This section presents relevant related work on data center networking in general and also specific to data intensive computing. Specifically, we summarize recent work in novel data center topologies, L2-, L3-, and L4-layer protocols, schedulers, resource allocators, traffic reduction, and rate controllers.

3.1 Topologies

Data intensive computing frameworks such as MapReduce require full-bisection bandwidth during the data shuffle phase. Unfortunately, a regular 2/3-layer tree topology is unable to cater to this need as links up the tree hierarchy are shared and hence, become a bottleneck. To remedy this, a number of novel topologies have recently been proposed.

Al-Fares et al.[13] present a data center architecture using a fat-tree topology which aims to support full-bisection bandwidth. A k-aray fat-tree enables each host and switch to have redundant links. IP addresses are assigned in the network based on the location of the element in the topology. To achieve even load balancing, a two-level routing table is employed. Further, a central route control statically generates all routing tables and loads them into switches. In addition, the architecture also enables flow classification and scheduling

to minimize overlap between flows. Finally, a failure broadcast protocol is used to enable switches to route around failures. VL2 [17] also uses a fat-tree topology but requires no changes to the underlying L2 fabric. It uses two different addressing schemes to simplify virtual machine migration. Interfaces and switches use location-specific addresses while application use application-specific addresses. The mapping between these two schemes is maintained by a replicated directory service. To spread traffic over the multiple links, VL2 uses both valiant load balancing (VLB) and equal-cost multi-path routing (ECMP).

In contrast to fat-tree and VL2, DCell [15] is a recursive, server-centric architecture. Each server is connected to other servers directly and through a small number of switches. DCells form a fully-connected graph and can be assembled together to form high-level DCells. Servers in DCells are identified by their position in the structure. Further, it uses a fault-tolerant single-path routing algorithm that exploits the recursive structure of the topology. A related architecture, BCube [16], is similar in structure but targets modular data centers [70]. Switches in BCube only connect servers, while servers themselves relay their traffic. Servers are addressed using an array based on its position, as a result neighbouring servers only differ by a single digit in the array. This information is used by source servers to encode routing information in the packet headers, which are in turn, used by servers for packet forwarding.

Another radical architecture is CamCube, which proposes a 3D torus topology in which each server is directly connected to a small set of other servers, inspired by structured overlays [49]. This scheme places each server into a 3D wrapped coordinate space and its position (x,y,z) in the space represents its address. As a result, the physical and virtual topology of the network is the same and applications can infer their location through an API which exposes coordinate space information and allows one-hop Ethernet packets. Thus, applications can implement their own routing strategies.

One problem that plagues these topologies is of incremental expansion. As a result of their rigid, well-defined structure, incremental expansion can only

be done if the structure is preserved. Jellyfish [24] addresses this problem by constructing a random graph topology. Random graphs achieve low diameter and high bisection bandwidth while allowing incremental expansion with any number of nodes. At the same time, Jellyfish can enable full-bisection bandwidth with a lower switch count than a fat-tree.

Finally, these topologies have also spawned a number of systems to improve their management and construction. Alias [71] and DAC [26] can be used to automatically assign addresses to these topologies. Perseus [25] is a framework that enables designers to choose an optimum data center design based on bandwidth, cost, application requirements, etc. The framework currently supports fat-tree and HyperX topologies. Specifically, user requirements and chosen topologies are given as input to Perseus which then generates a design with a candidate topology, optimized cabling, cost, and a visualization of the structure.

Although, these topologies expose full bisection bandwidth, in practice due to multi-tenancy [72], and TCP's inefficiency in (a) moving away from congested links, and (b) exploiting multiple links [73], applications do not experience the full benefit. In addition, these topologies also increase the overall cost and wiring complexity. In data centers which already have these topologies, Mission Control can be used to exploit their multiple links.

3.2 Ethernet Fabrics

Ethernet also faces a number of scalability challenges in a data center environment.

SPAIN [18] is an Ethernet fabric that implements forwarding along multiple paths connected by commodity switches and arbitrary topologies. Specifically, it overcomes the single spanning restriction of the Spanning Tree Protocol (STP) by making use of multiple spanning trees which are calculated offline. In topologies with multiple paths, each spanning tree is organized into a separate VLAN to exploit path redundancy. In addition, it also min-

imizes the overhead of packet flooding and host broadcasts. NetLord [28] is a related system which enables flexibility in choosing VM addresses. To this end, it implements IP encapsulation for custom L2 packets and uses SPAIN as the underlying fabric. This enables NetLord to scale to a large number of tenants and VMs. It also uses a custom push-based ARP protocol to keep location information consistent. Similarly, Portland [27] is a layer 2 routing and forwarding fabric that uses topological information to assign end-hosts hierarchical pseudo-MAC (PMAC) addresses. It employs a centralized directory service to map PMAC onto physical MAC addresses. All routing and forwarding takes place using PMAC prefixes while the end-hosts only see MAC addresses. In addition, the centralized directory service eliminates the need for broadcast based ARP and DHCP. Finally, end-hosts are modified to spread load over the available paths. Vattikonda et al. explore a new point in the design space by dispensing with TCP altogether and replacing it with a time-division multiple access (TDMA) MAC layer [19]. To this end, they employ centralized scheduling to give communicating end-hosts exclusive access to links. This exclusive assignment of link bandwidth and switch buffer space automatically mitigates in-network queuing and congestion. All of these proposals work at layer 2 and are as such, complementary to our work.

3.3 Global Schedulers and Resource Allocators

To enable applications to make use of multiple paths in the network, a number of schedulers have also been proposed. One such scheduler is Hedera [33] which uses flow information from Openflow switches to compute non-conflicting paths for flows, which are in turn pushed back to the switches. This enables the network to maximize bisection bandwidth. It provides a number of algorithms for choosing optimum paths. Similarly, DevoFlow [74] is a modification of the Openflow architecture that tries to handle most flows

in the data plane to achieve scalability. Hence, Hedera-like schedulers only need to deal with *interesting* flows.

MicroTE [48] is a traffic engineering systems that adapts to variation in network traffic at the granularity of a second. It consists of a central controller that creates a global view of traffic demands and coordinates scheduling using Openflow. To this end, it differentiates between predictable traffic and non-predictable traffic. For the former, it computes routes based on some global objective to achieve optimum routing. The leftover routes are then used to stripe unpredictable traffic using weighted equal-cost multi-path routing (ECMP).

A recent system, Orchestra [37], works at the application layer to enable intra- and inter-transfer control of two communication patterns, which lie at the heart of batch processing systems such as MapReduce: *shuffles*, and *broadcasts*. For the former, it uses weighted flow assignment (WFA) enabled by an appropriate number of multiple TCP connections and TCP's AIMD fair sharing while for the latter it leverages a BitTorrent-like algorithm. Architecturally, it is a centralized application layer controller for both intra- and inter-transfer communication. The former is managed by one or more Transfer Controllers (TCs) while the latter is managed by an Inter-transfer Controller (ITC). The TC continuously monitors the transfer and updates the set of sources associated with each destination. Specifically, the TC manages the transfer at the granularity of flows, by choosing how many concurrent flows to open from each node, which destinations to open them to, and when to move each chunk of data. While the results from Orchestra are promising, it is still suboptimal due to its reliance on TCP. In addition, frameworks like Dryad support TCP pipes, and shared memory FIFO for communication in addition to transfer of files over the network, while Orchestra only considers the last of these.

In contrast to all of these systems, Mission Control uses the diversity in transport protocols to achieve optimum routing and congestion control. At the same time, the architecture of Mission Control is inspired by the architecture of Orchestra.

3.4 Transport Protocols

In this section we enumerate transport protocols which address some of TCP's shortcomings.

Deadline-Driven Delivery (D^3) [22] is a data center-specific version of TCP that targets applications with distributed workflow and latency targets. These applications associate a deadline with each network flow and the flow is only useful if the deadline is met[1]. To this end, applications expose flow deadline and size information which is exploited by end hosts to request rates from routers along the data path. Routers only maintain aggregate counters and thus, do not maintain any per-flow state. As a result, they are able to allocate sending rates to flows in order for as many deadlines to be met as possible. On the downside, D^3 requires changes to applications, end-hosts, and network elements. For data centers with customizable network elements, Mission Control can use D^3 for applications that expose explicit deadline information.

Data Center TCP (DCTCP) [14] uses Explicit Congestion Notifications (ECN) from switches to perform active queue management-based congestion control. Specifically, it uses the congestion experienced flag in packets—set by switches whenever the buffer occupancy exceeds a small threshold—to reduce the size of the window based on a fraction of the marked packets. This enables DCTCP to react quickly to queue buildup and avoid buffer pressure. DCTCP can only work in a network with ECN-compatible switches. Therefore, Mission Control enables DCTCP if it detects that switches in the network support ECN. A spin-off architecture is HULL [75], high-bandwidth ultra-low latency, that aims to make a trade-off between bandwidth and latency for low-latency applications in a shared environment. To this end, it reduces network buffering by replacing queue occupancy-based congestion control with link utilization. It makes use of *phantom queues* which give the illusion of draining a link at less than its actual rate. End-hosts employ-

[1] TCP on the other hand tries to achieve global fairness and maximize throughput while remaining agnostic to any flow-specific deadline.

ing DCTCP experience reduced transmission rate based on ECN from these phantom queues. This enables the overall network to trade bandwidth for latency.

Structured Stream Transport (SST) [76] finds the middle-ground between stream and datagram transmission. It implements a hierarchical and hereditary stream structure which allows applications to create substreams from existing streams without incurring startup and tear-down delay. Substreams operate independently with their own data transfer and flow control. To ensure inter-stream fairness, substreams share congestion control. Moreover, "ephemeral streams" can be used to enable datagram transmission. Finally, applications can prioritize their streams on the fly depending on runtime requirements. Mission Control can use SST to enable MapReduce-like systems to prioritize their data transfers, by opening multiple substreams, and to send lightweight control traffic using ephemeral streams.

Raiciu et al.[21] argue that current data center architectures use random load balancing (RLB) to balance load across multiple paths. RLB is unable to achieve full bisectional bandwidth because certain flows always traverse the same path. This creates hotspots in the topology, where certain links are oversubscribed and some are under/un-subscribed. Global flow controllers fail in this regard too as they are unable to scale with the number of flows. To remedy this, they propose the use of multipath TCP (MPTCP) to establish multiple subflows over different paths between a pair of end-hosts. The key point is that these subflows operate under a single TCP connection. They argue that this ensures fairness as the fraction of the total congestion window for each flow is determined by its speed, i.e. the faster a flow is, the steeper the slope of its additive increase. This will also move traffic away from the most congested paths. MPTCP is supposed by Mission Control in multipath networks.

3.4.1 Incast

A number of proposals have also tried to tackle incast.

Vasudevan et al. [20] propose that we increase the granularity of kernel timers to enable microsecond granularity RTT estimation, which is in turn used to set the RTO. They show that reducing the RTO can increase the goodput of simultaneous connections and hence negate incast. Furthermore, making retransmissions random can also increase the goodput by stopping multiple flows from timing out, backing off, and retransmitting simultaneously. The takeaway lesson is that to avoid incast in low-latency data center networks, RTOs should be on the same scale as network latency. Chen et al. [35] complement the analysis of the previous work by defining an analytical model for incast. This model shows that goodput is affected by both the minimum RTO timer value and the inter-packet wait time. Unlike these two solutions which focus on post hoc recovery, ICTCP [23] aims to avoid packet losses before incast manifestation. To this end, ICTCP implements a window-based congestion control algorithm at the receiver side. Specifically, the available bandwidth at the receiver is used as a trigger to perform congestion control. The frequency of this control changes dynamically based on the queueing delay and is larger than one RTT. Connection throughput is only throttled to avoid incast, not to decrease application goodput.

3.5 Data Center Traffic Analysis

Kandula et al. [36] present an analysis of the traffic of a 1500 server data center. They use lightweight measurements at end-hosts in conjunction with application level data to note the cause and effect of network incidents. The target applications have a work-seeks-bandwidth and scatter-gather pattern. Their analysis shows that most traffic is exchanged within a rack and on average, a server communicates with two servers within its rack and four outside the rack. With respect to congestion, their study shows that highly utilized links are prevalent and congestion can last from 10s of seconds to 100s. In addition, most periods of congestion are short lived. Further, during periods of congestion there is an increase in the number of worker failures. Finally, traffic patterns change both frequently and quickly over time. Fol-

lowing in the footsteps of Kandula et al., Benson et al. [77], present SNMP data from 19 data centers. Their analysis shows that traffic load is high in the core and decreases towards the edge. At the same time, link loses are higher near the edge and decrease towards the core. In addition, a small fraction of the links account for most losses. This observation shows that traffic can be routed along alternate routes to avoid congestion. Finally, traffic is bursty in nature and packet inter-arrival times during spike periods follow a lognormal distribution. In a follow-up paper, Benson et al. [7] augment their previous work by studying different classes of data centers. These include, university campus, private enterprise, and cloud data centers. In this work, they examine the variation in link utilization and the magnitude of losses as well as the nature of hotspots. Their measurements show that for cloud data centers, 80% of the traffic stays within the rack and the number of highly utilized core links never exceeds 25% of the total links.

We use the insight provided by these studies to keep the design of Mission Control generic enough to cater to the needs of these widely-varying traffic patterns.

3.6 Reducing Network Traffic

Another point in the design space is to reduce the amount of traffic that is transferred across the network by MapReduce-like systems [72, 78]. This can be done either across different levels in the cluster topology [78], or made a function of the network itself as in Camdoop [72]. On the downside, performing rack-level aggregation can saturate the ingress link to the aggregating server [78]. This *distributed combiner* approach is most effective for MapReduce functions which are commutative and associative [4]. In the same vein, SUDO [79] leverages the functional and data-partition properties of user-defined functions in MapReduce-like systems to enable various optimizations. This enables the framework to avoid unnecessary data-shuffling steps and reduce disk and network I/O. While these frameworks reduce the

amount of data transferred across the network, they still rely on standard TCP, inheriting its flaws.

3.7 Wireless and Optical Networks

Another line of work has argued that providing full-bisection links throughout the network is overkill as only a few hotspots exist in actual data centers [10, 39]. The high cost and technical challenges of newer topologies can be avoided by provisioning the network for the average case and avoiding hotspots by adding additional links on the fly. The key challenge is to choose the placement and duration of *flyways*. In these proposals, flyways are constructed using Multi-Gigabit wireless links, controlled by a centralized scheduler that decides the temporal and spatial placement of wireless links based on observed traffic. On major shortcoming of this approach is that in cases where the traffic has high churn, the controller would fail to respond quickly.

Likewise, c-Through [11] and Helios [12] consider using hybrid packet and circuit switched networks in order to use optical links to address hotspots. c-Through connects racks using optical circuit-switched links based on traffic demand. Network isolation is ensured by using a separate VLAN for the optical and electrical rack links. End-hosts are used for traffic statistics collection while a central manager, connected to the optical switch, is used to manage configuration. In contrast, Helios [12] targets modular data centers [70]. It uses flow counters in switches to calculate the traffic matrix and in turn, uses that information to work out a topology to maximize throughput. Finally, OSA [80] is a clean-slate all-optical network which can alter its topology and link capacities based on the traffic demand.

While we do not consider wireless and optical links in this work, Mission Control is flexible enough to support both. For example, Mission Control maintains a global view of the network, therefore it can choose flyways or optical paths for hotspots, if required.

3.8 Virtualization and Rate Limiting

To take advantage of economies of scale and achieve optimum resource utilization, most data centers are virtualized. As a result, resources are shared amongst the tenants, which can lead to contention and other undesired behaviour.

Due to VM consolidation, the RTT of TCP connections to VMs increases significantly, causing the latency of connections to go up. vSnoop is a system that allows the driver domain of a host to send an acknowledgement on behalf of the VMs [42]. It snoops on all VM traffic and maintains per-flow state for each TCP connection which it then uses to enable early acknowledgement of packets. The same problem afflicts the transmit path as well, as TCP acknowledgements can get delayed, causing the TCP congestion window to grow more slowly. To mitigate this, vFlood is another extension that offloads congestion control to the driver domain, which allows the buffering of a high number of packets [41]. Regardless of the transport protocol, Mission Control can delegate congestion control to the driver domain in virtualized environments.

Another related body of work deals with ensuring bandwidth fairness among virtual machines and tenants in the data center. Seawall [81] uses hypervisor-based rate limiting in conjunction with IP layer feedback signals from the network to regulate egress traffic. Standard rate control algorithms, such as TCP, TCP friendly rate control (TFRC), or quantized congestion notification (QCN) are used to implement rate limiting. On the downside, as Seawall is VM-centric, it does not divide the link bandwidth among tenants using the link, but among the total number of VMs sending traffic through that link. Therefore, tenants with a large number of VMs unfairly get a large share in bandwidth. Similarly, Gatekeeper [32] works at the virtualization layer of end-hosts to ensure bandwidth fairness. It achieves scalability by using a point-to-point protocol and maintains minimum data center-wide control state. Maximum and minimum rate is set for both ingress and egress traffic at the vNIC level and scheduled using a weighted fair scheduler. In

addition, ingress and egress traffic is monitored and if limits are exceeded, a feedback message with an explicit rate is sent to all VMs involved in the traffic. Likewise, SecondNet [29] uses a *virtual data center* (VDC) abstraction for resource allocation. It allows the assignment of a set of VMs with computation, storage, and bandwidth guarantees. Architecturally, a central manager is in charge of all resources while virtual-to-physical mappings and network state is controlled by the source hypervisors. While SecondNet can guarantee end-to-end bandwidth, it can only be used if the bandwidth requirements are known a priori. Choosing an exact bandwidth is non-trivial for most applications as bandwidth is a function of their communication pattern. Oktupus [30] is a resource allocator that uses "virtual networks" as an abstraction to expose to users. This allows the cloud provider to decouple the virtual network and the underlying physical network. As a result, there is a trade-off between application goals for performance and provider goals for optimum utilization of resources. For all-to-all communication, such as MapReduce traffic, a *virtual cluster* is employed which provides the illusion of all virtual machines being connected to the same non-oversubscribed virtual switch. For applications with a local traffic pattern, a *virtual oversubscribed cluster* is employed which emulates an oversubscribed two-tier cluster. Specifically, Oktupus uses physical network topology, machine usage, etc., to allocate physical machines to users. In addition, explicit bandwidth rate-limiting at the end hosts is employed to control bandwidth.

In this work we do not explicitly tackle VM rate limiting, but Mission Control as an architecture can be used to rate limit ingress and egress links at the application layer by controlling the transport schedule.

3.9 Energy Efficiency and Security

In addition to performance and fairness, energy efficiency and security are of great important in the data center as well.

ElasticTree is a system that dynamically alters the power consumption of

networking elements inside a data center based on the current traffic [50]. It consists of an optimizer, routing control, and power control. Based on the topology, traffic matrix, and other inputs, the optimizer computes an optimum power network subnet which is in turn fed to the power control and the routing control. The former toggles the power state of network elements, and the latter chooses paths for each flow. The optimizer provides a number of strategies which make a trade-off between scalability and optimality. ElasticTree can be used in conjunction with Mission Control to choose a suitable underlying transport protocol. For example, the performance of TCP is adversely affected if there is packet reordering due to splitting a single flow across multiple links by ElasticTree.

TCPcrypt is a backwards compatible enhancement to TCP that aims to efficiently provide encrypted communication transparently to applications [51]. To this end, it uses a custom key exchange protocol that leverages the TCP options field. Like SSL, to reduce the cost of connection setup for short-lived flows, it enables cryptographic state from one TCP connection to bootstrap subsequent ones. Even for legacy applications, it protects against passive eavesdropping. Additionally, TCPcrypt protects the integrity of TCP sessions, and defends against attacks that change their state or affect their performance. Finally, applications can also be made aware of the presence of TCPcrypt to negate redundant encryption. We use TCPcrypt as a secure transport protocol option in Mission Control.

In summary, novel data center topologies in practice do no achieve full-bisection bandwidth due to their reliance on TCP. In fact, in some cases full-bisection bandwidth is overkill and the needs of most applications can be satisfied by using a few additional links. Extensions to Ethernet can address its scalability problems in the data center but the transport is still at the mercy of TCP. In addition, global schedulers can take advantage of multiple paths to improve performance but they either use TCP or ECMP which are by nature suboptimal in the data center. Moreover, novel transport protocols only work in specific environments for a limited number of traffic patterns. Furthermore, aggressive use of *combiners* can only reduce traffic to

a certain extent. Finally, rate limiters expect users to know the bandwidth requirements of their applications beforehand. This is impractical as the prediction of network traffic is non-trivial and application requirements are dynamic over the course of their execution. Mission Control addresses the shortcomings of all of these systems by enabling diversity in both transport semantics and scheduling.

Chapter 4

Design and Implementation

In this section, we present the design and implementation of Mission Control.

We first enumerate a set of goals for Mission Control:

1. Mission Control should work out of the box. Therefore, it should require no changes to network infrastructure.

2. There is no "one-size-fits-all" transport for the data center. Mission Control should support a range of diverse transports.

3. Data intensive computing frameworks, such as MapReduce require minimal intervention from the user. In fact, these systems have found traction because they can run unmodified user binaries. Mission Control should not modify the programming model of these frameworks in any way.

4. The architecture of MapReduce-like systems consists of loose coupling between the master and workers. As a result, these systems achieve fault-tolerance, freedom from side-effects, and simple coordination. Mission Control should also work on the same principles.

5. Data intensive computing frameworks these days scale to thousands of worker nodes. In addition, their architecture enables them to seamlessly, scale-up and scale-down. Therefore, Mission Control should not

limit the scalability of these systems.

6. Like mentioned earlier, bandwidth is a constrained resource in the data center. Mission Control should only add minimal control traffic to the mix. To this end, it should make use of multicast whenever possible.

4.1 Design

In light of the goals presented in the previous section, we now describe the design of Mission Control.

Architecturally, Mission Control consists of three key components: (1) Mission Controller (MC), (2) Flight Controller (FC), and (3) Throttle Controller (TC). These components are described in detail below.

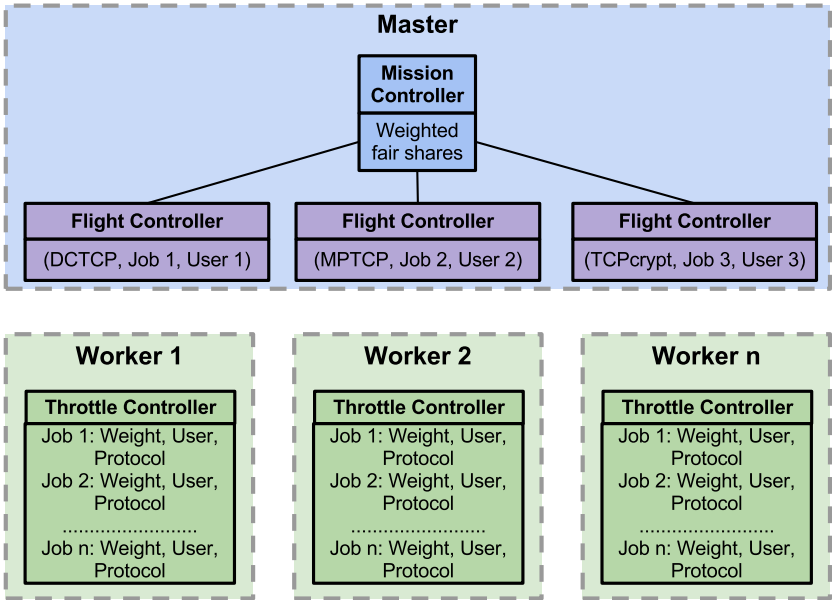


Figure 4.1: Mission Control architecture

4.1.1 Mission Controller

CIEL is capable of running multiple jobs at the same time. Mission Controller resides alongside the CIEL master and coordinates transfer scheduling across these jobs. To this end, it uses *weighted fair sharing* to assign each transfer a weight. In practice, this allows the sharing of each link in the network proportional to these weights. In addition, weighted fair sharing subsumes several policies including FIFO and priority [37]. The values of these weights are subject to organizational goals and can be calibrated on the basis of user profiles or job characteristics. For example, jobs launched by *Alice* can be given twice the priority of those launched by *Bob*. In the same vein, low latency jobs can be given higher priority than batch-processing jobs. Similarly, to enable FIFO, weights can be assigned proportional to job arrival time. By default we use performance-centric network allocation to maximize parallelism [82]. For shuffle transfers allocation is made on the basis of the number of target machines while per-flow allocation is used for broadcast transfers. Concretely,

$$share_{shuffle} = N \tag{4.1}$$

$$share_{broadcast} = N^2 \tag{4.2}$$

Where N is the number of machines that the job uses. To distinguish between shuffle and broadcast transfers, Mission Controller walks the task graph to observe the relationships between tasks and their inputs and outputs (described in the implementation section). In addition, transport protocols are selected on the basis of job characteristics and the environment. For example, multi-path TCP is selected if the host has more than one interface. Moreover, users can explicitly specify a transport protocol at job submission time. In summary, Mission Controller maintains cluster-wide state of resources and selects the transport layer protocol for a particular transfer.

4.1.2 Flight Controller

Each transfer is associated with a Flight Controller that handles transport specific resources. To this end, each FC implements the control plane of a particular transport protocol and its various parameters. For example, the multipath TCP FC selects the number of flows per TCP connection. Each FC is registered with the Mission Controller and receives its weight assignment. It is important to highlight that to change the transport protocol at runtime, the Mission Controller unregisters the existing FC for a transfer and replaces it with a corresponding FC for the new protocol.

4.1.3 Throttle Controller

Unlike the Mission Controller and Flight Controller(s), which reside on the master node, Throttle Controllers are present at every worker. Each TC receives its transfer schedule from the FC and uses that information to handle all flows on the worker. Each TC receives information for each FC (or job) for which it has to schedule a transfer (or task). As a result, Throttle Controllers have to maintain both scheduling and transport parameter state for each local flow. Figure 4.1 illustrates the architecture of Mission Control.

4.2 Implementation

In this section, we present how Mission Control extends the implementation of CIEL to enable multiple transports and transfer scheduling. We first discuss the implementation features of CIEL which are important for Mission Control.

Reference	Description
Future	Represents an object that has not been created yet
Concrete	For objects that have concrete values and locations
Value	Stores the actual value of small objects
Error	Stores the error code of failed tasks
Streaming	Enables streaming data between tasks
Sweetheart	Represents loop-invariant data
Tombstone	Represents objects that are no longer available due to worker failure
Fixed	Enables a task to be fixed to a worker

Table 4.1: CIEL References

4.2.1 CIEL Implementation

CIEL represents the state of each job in a *dynamic task graph* which records the relation between tasks and objects in the form of a bipartite graph [6, 83]. The former is stored in a task table while the latter is stored in an object table. The task table stores the code and data dependencies of each task keyed by task ID. Similarly, the object table stores the input and output objects for each task keyed by object ID. All of this information is stored in the form of *references*. CIEL enables a number of references which are described in Table 4.1. The master stores a table of dynamic task graphs with one entry for each job in the job pool. In addition, worker state is stored in a worker table.

Tasks are scheduled on workers from the worker table once their dependencies, in the form of future references, become concrete references. The task table is updated each time a new task is spawned. Moreover, objects are stored within an *object store* at each worker. All of this information is leveraged by Mission Control to optimize data transfer (described below). This data-dependent abstraction can be used to implement any distributed data flow. For instance, a fixed two-stage MapReduce flow can be enabled by adding one output Reference from each Map task as a dependency to each Reduce task.

As mentioned earlier, tasks are scheduled once their dependencies become

concrete. To facilitate this, each *concrete reference* consists of a name, location and size. Objects are retrieved by sending an HTTP GET request with the name of the object to the location[1] specified in the concrete reference. Under the hood, PycURL[2], a Python interface to libcURL[3] is used to fetch the object. libcURL is a URL transfer library that supports multiple protocols including HTTP and FTP. Furthermore, CIEL also allows partially written objects to be pipelined between tasks through *stream references*. Finally, CIEL exposes a language-agnostic interface that can be used to spawn tasks in any programming language, including Skywriting, Python, Java, and C. This *first-class executor* enables a running task to, (1) Spawn other tasks, (2) Delegate its output(s), and (3) Create new objects on the fly. In this thesis, we focus on the Java first-class executor, but the design is applicable to executors in any language.

Listing 4.1: `FirstClassJavaTask` Interface

```
1  public interface FirstClassJavaTask extends Serializable
2  {
3      /*
4       * Enables the task to setup any local state
5       */
6       void setup();
7
8      /*
9       * Holds the actual code for execution
10      */
11      void invoke() throws Exception;
12
13     /*
14      * Returns an array of the dependencies of the task
15      */
```

[1] In case the reference has multiple locations, the request is directed on the basis of network locality or at random.

[2] `http://pycurl.sourceforge.net/`

[3] `http://curl.haxx.se/libcurl/`

Method	Description
`Ciel.args`	Returns user-supplied arguments
`Ciel.spawn`	Spawns a task with the given code dependency
`Ciel.tailSpawn`	Spawns a task with the given code dependency and delegates the expected output(s) of the current task to it
`Ciel.RPC.getStreamForReference`	Opens the object defined in the reference in pipelining mode
`Ciel.RPC.getFilenameForReference`	Fetches the object defined in the reference
`Ciel.RPC.getOutputFilename`	Creates a new reference to hold the output(s) of the task

Table 4.2: Important CIEL First-class Java Executor helper methods

```
16      Reference[] getDependencies();
17  }
```

Java First-class Executor

Tasks in Java can be written by implementing the `FirstClassJavaTask` interface shown in Listing 4.1. In addition, the platform also provides a `ConstantNumOutputsTask` interface which extends `FirstClassJavaTask` with a `getNumOutputs` method to hold the number of outputs of a task. Finally, tasks also have access to a `Ciel` object which exposes a number of helper methods, some of which are described in Table 4.2.

4.2.2 Mission Control Implementation

Mission Control extends both the Python implementation of CIEL and its first-class Java executor. The former[4] is modified to implement Mission Controller, Flight Controller, and Throttle Controller while the latter[5] is

[4] `https://github.com/ZubairNabi/ciel`
[5] `https://github.com/ZubairNabi/ciel-java`

Policy	Description
FIFO	Jobs are assigned weights on the basis of their arrival time.
Priority	Job priorities are specified by the user. A job with a priority of 2 is assigned a weight which is double the weight assigned to a priority 1 job.
Fair	Jobs are assigned equal weights.
Performance-centric	Weights are assigned to maximize performance parallelism. A broadcast transfer is assigned twice the weight of a shuffle transfer.

Table 4.3: List of weight assignment policies supported by Mission Control

extended to enable concurrent data fetch threads. In addition, a number of Bash scripts[6] have also been written to manage CIEL and the working environment. We first describe the changes to the master followed by changes to the worker.

Master

At the master, the Mission Control implementation consists of two classes to represent Mission Controller and Flight Controller, respectively. We modified the CIEL representation of a job to associate each job with a weight and a Flight Controller. The Mission Controller thread is spawned during the initialization of the master. It maintains an updated list of currently running jobs along with their tasks and dependencies. In addition, it is also associated with a policy which dictates per-job weight assignment. The list of currently supported policies is given in Table 4.3. The weights assigned by these policies determine the number of concurrent flows that each job can initiate during a data fetch. These weights are updated every second

[6] `https://github.com/ZubairNabi/ciel/tree/master/scripts`

Policy	**Weights**
FIFO	As soon as Job 2 arrives, it is assigned a weight of 3 and the weight of Job 1 is decreased to 7.
Priority	Job 2 arrives and has a higher priority than Job 1. Job 2 is assigned a weight of 7 and the weight of Job 1 is decreased to 3.
Fair	Job 2 arrives and is assigned a weight of 5 and the weight of Job 1 is decreased to 5.
Performance-centric	Job 1 is has a shuffle transfer while Job 2 has a broadcast transfer. Job 2 arrives and is assigned a weight of 7 and the weight of Job 1 is decreased to 3.

Table 4.4: Example of weight assignment under different policies for 2 jobs

depending on the selected policy. Table 4.4 shows how weights are assigned under different policies for 2 jobs: Job 1 and Job 2. The maximum number of concurrent flows per host is 10 and Job 1 arrives first and is assigned all 10 flows.

Be default, Mission Controller uses the performance-centric [82] policy in which weights are assigned on the basis of application semantics. To this end, Mission Controller performs a breadth-first search of the dynamic task graph for each job. If the tasks in a stage have one reference from each task in the previous stage as a dependency, the stage is marked as a shuffle. In contrast, if all of the output references of the previous stage are dependencies of the subsequent stage, then the stage is marked as a broadcast. This information is then used to assign twice the weight of a shuffle transfer to a broadcast transfer. In addition, the current implementation supports four transport protocols. These protocols and their selection criteria is given in Table 4.5.

Protocol	Description
TCP	The default transport.
MPTCP	Selected when the host has more than one interface. In case MPTCP is selected, the number of concurrent data fetch connections per host are turned into concurrent sub-flows under the same MPTCP connection.
DCTCP	Selected when the switches in the network support ECN.
TCPcrypt	Not selected by the framework. It needs to be explicitly enabled during job submission time by the user.

Table 4.5: List of transport protocols supported by Mission Control

It is important to highlight that any transport protocol can easily be added to Mission Controller by extending the Flight Controller class.

Different Kernels, Isis2, Python, C#, and IronPython[7].

Isis2 [84][8] is a toolkit which implements the virtual synchrony model and simplifies the design of distributed applications. It provides a number of reliable multicast primitives with varying ordering guarantees. Durability is ensured through a mechanism dubbed, *amnesia freedom*, that delays the application till an acknowledgement has been received from every replica. This mechanism is exposed through a barrier primitive called *Flush*, that delays till all prior multicast messages have reached their destinations. Isis2 also provides a number of send primitives including a FIFO-ordered send as well as a *SafeSend*, which implements a virtually synchronous version of Paxos [85].

[7] `http://ironpython.net/`
[8] `http://isis2.codeplex.com/`

Worker

At the worker, the implementation consists of a Throttle Controller instance that receives data transfer semantics from the Mission Controller. In particular, it distributes its total number of flows among the executing tasks. This number is bounded by `pycurl.M_MAXCONNECTS` at ever host. In CIEL, executors in different programming languages communicate with the underlying framework using JavaScript Object Notation (JSON)[9] based RPC. For instance, the methods in Table 4.2 are implemented through this interface. To enable the Throttle Control to allocate concurrent flows to each executing task, the RPC methods are extended to include a `Ciel.RPC.getConcurrentFilenamesForReferences` method to spawn multiple threads to make concurrent calls to the built-in `Ciel.RPC.getFilenameForReference` method. The number of these threads per task is equal to its concurrent flow quota. As a result, the Throttle Controller is able to distribute its `pycurl.M_MAXCONNECTS` number of concurrent flows among the executing tasks. Moreover, on instructions from the Mission Controller, the Throttle Controller switches between transport protocols using the Linux `sysctl`[10] interface.

[9] `http://www.json.org/`

[10] `sysctl` is used to modify kernel parameters at runtime.

Chapter 5

Evaluation

This section presents a thorough evaluation of Mission Control. We first describe the testbed.

5.1 Testbed

For all our experiments we use an AMD Opteron 6168 with 48 x86_64 cores, with each core clocked at 2 GHz with a 512 KB cache. The system has 64 GB of DDR3 RAM, 1 TB of HDD, and 1 Gbps network connectivity. In addition, the system is also provisioned with a 256 GB solid state drive[1].

On the software side, the system is running a Debian-based Linux kernel version 3.2.x.x. We use Linux Containers (LXC)[2] for lightweight virtualization. 41 containers were created to host the CIEL[3] master and workers. The container housing the CIEL master was pinned to 4 CPU cores while the rest of the 40 containers, running CIEL workers, were pinned to 1 CPU core each. In addition, the former was provisioned with 100 GB of HDD space in the form of a logical volume while the latter were provisioned with 20 GB of

[1] We use the SSD to negate the effect of disk I/O on shuffle time.
[2] `http://lxc.sourceforge.net/`
[3] `https://github.com/mrry/ciel`

the same each. The SSD is shared by all. Finally, OpenVSwitch [86][4] is used as the L2 fabric with 1 Gbps link speed.

For all our experiments, we use a custom MapReduce Java library written for CIEL[5]. This enables us to (a) Easily port Hadoop applications to CIEL, (b) Monitor and log phase-wise task execution. We implement different topologies by connecting and rate-limiting OpenVSwitch-enabled bridges. To monitor the CPU and network usage of each worker, we poll the `/proc` filesystem. Unless otherwise stated, all application data (input, intermediate, and output) is stored on the SSD. In addition, all error bars represent the standard deviation.

5.2 Analysis of network behaviour of CIEL on different workloads

To gain some insight into the network behaviour of CIEL, we first conduct a series of experiments using an I/O-intensive and a CPU-intensive application. For the former we use sort and for the latter we use k-means clustering.

5.2.1 Sort

In our sort implementation, the map and reduce functions are identity while the underlying framework itself performs the actual sort. In this experiment, our goal is to study the effect of data size on the shuffle and the overall job completion time. A 20 GB text dataset was synthetically generated to ensure equal partitioning and equal distribution of keys, i.e. task times are not biased by unequal work distribution. Each file in the dataset is 64 GB in size, with 640 files in total. Each file is given as input to a single map

[4] `http://openvswitch.org/download/`

[5] `https://github.com/ZubairNabi/ciel-java/tree/master/examples/src/main/java/com/asgow/ciel/examples/mapreduce`

task while the number of reduce tasks is kept constant at 40[6]. Each CIEL worker is configured for a single task slot. In this setup, all 41 containers are connected to the same switch.

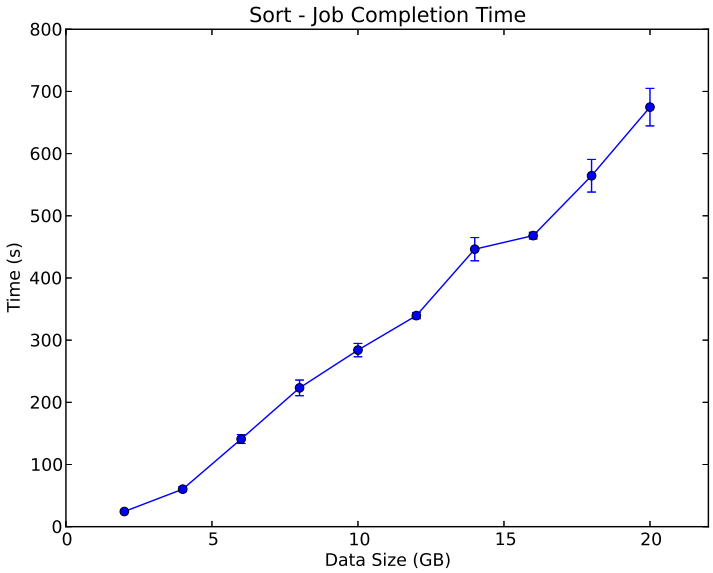


Figure 5.1: Sort - Job Completion Time

Figure 5.1 shows the results of 10 runs of sort for input data size varied from 2 GB to 20 GB. The job completion time linearly increases with the increase in input. A phase-wide breakdown of the job completion time in Figure 5.2 reveals that the shuffle phase accounts for up to half of the total time.

To understand the effect of the application on system resources, we plot the CPU and network usage of sort for input data of size 2, 10, and 20 GB in Figure 5.3. We see that there is a spike in network traffic during the shuffle phase while otherwise the network is under-utilized.

[6]One per CIEL worker.

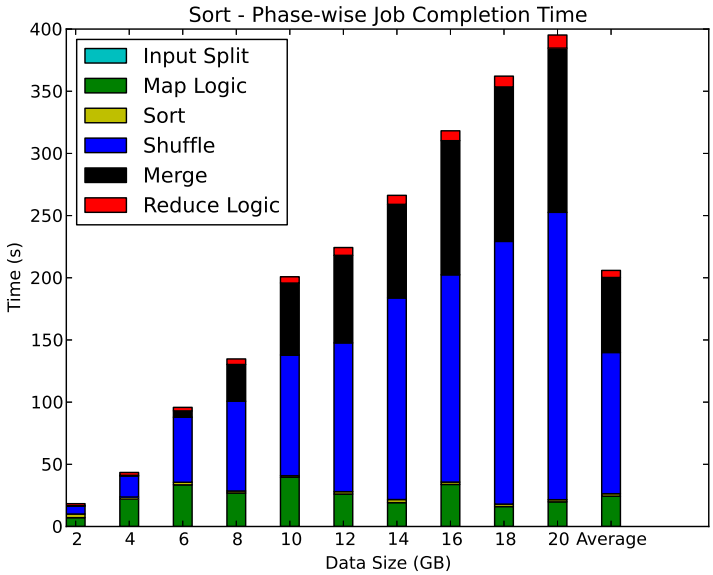
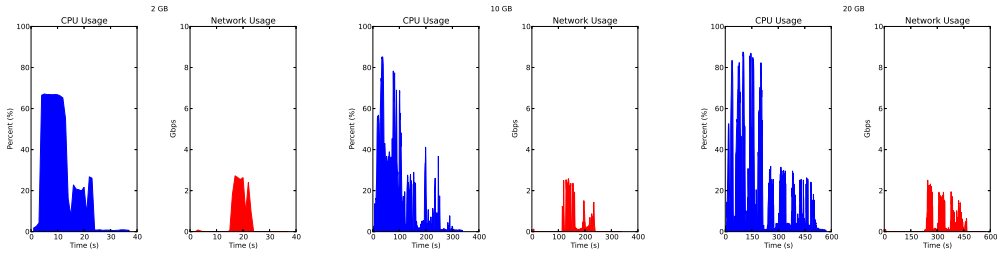


Figure 5.2: Sort - Phase-wise Job Completion Time



(a) 2 GB - 32 maps (b) 10 GB - 160 maps (c) 20 GB - 320 maps

Figure 5.3: CPU and Network usage of sort

5.2.2 *k*-means clustering

We now consider k-means clustering, a CPU intensive application. This application takes advantage of CIEL's dynamic task generation by iterating till the result is within an acceptable value. The CIEL port of this application is identical to its Mahout[7] implementation without requiring an external driver. The map function assigns each input point to its nearest cluster based on a distance metric. The reduce function sums up the values and

[7] `http://mahout.apache.org/`

checks if they have converged. In case they have not, it launches the next iteration of the application. To evaluate this application, we generated 64 MB cluster files with 80,000 dense vectors and k, cluster center, value of 100. Each vector contains 100 double-precision values [6].

Figure 5.4 plots the job completion time of k-means as a function of the input data size. In contrast to sort, the execution time of k-means shoots up with an increase in input. This is due to the CPU intensive nature of the application as well as the dynamic task generation. The CPU and network usage of the application is shown in Figure 5.5. Again in contrast to sort, we see network and CPU activity throughout the course of the execution.

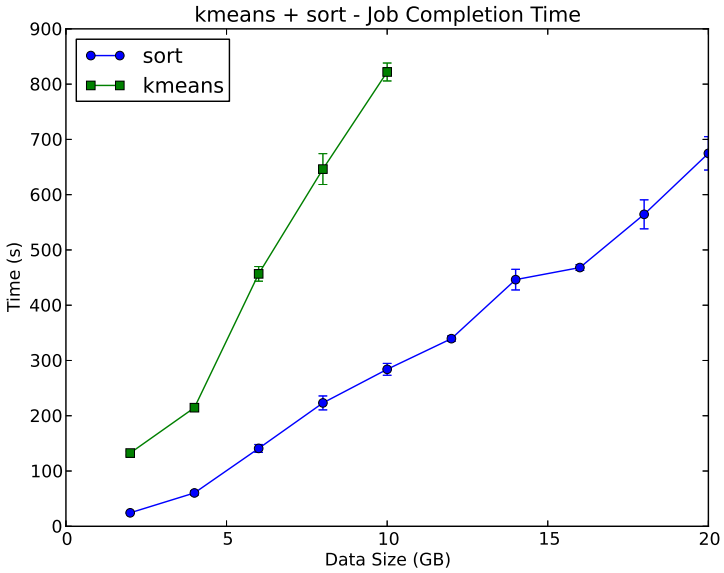
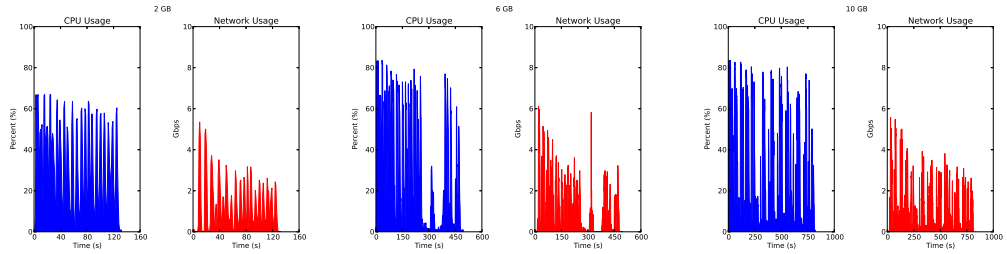


Figure 5.4: Sort and k-means Job Completion Time

5.3 Performance of CIEL using different transport semantics

In this section we evaluate the performance of CIEL under different transport semantics.



(a) 2 GB (b) 6 GB (c) 10 GB

Figure 5.5: CPU and Network usage of k-means

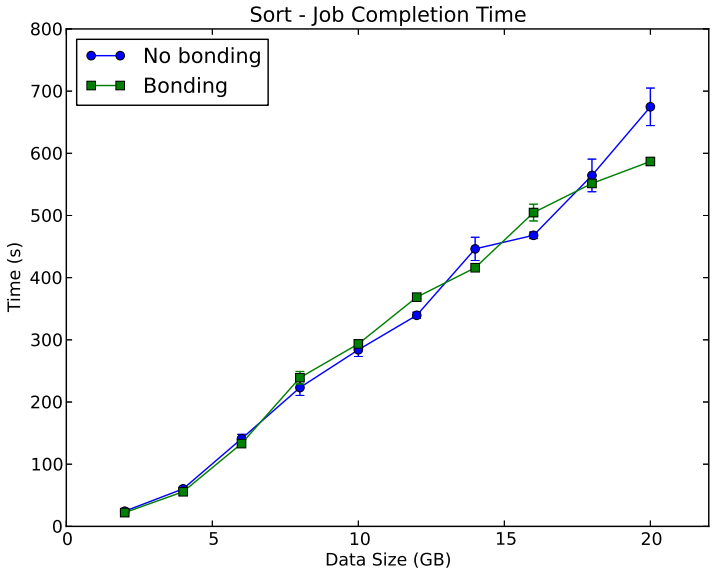


Figure 5.6: Sort with Bonding - Job Completion Time - Star topology

5.3.1 Ethernet Bonding

We first evaluate Ethernet bonding. We add an extra interface to each container and use the `bonding` module in Linux to bond both interfaces per container. To stripe traffic from the same TCP connection over the interfaces, we make use of the `balance-rr` bonding mode. Figure 5.6 shows the job completion time of CIEL sort with and without Ethernet bonding. Both methods exhibit the same behaviour.

To gain more insight, in Figure 5.7 we only focus on the shuffle time. On

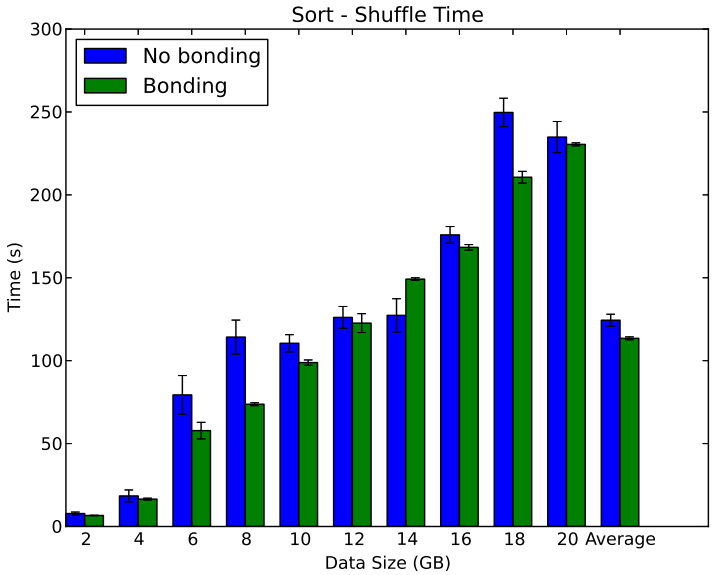


Figure 5.7: Sort with Bonding - Shuffle Time - Star topology

average, Ethernet bonding improves shuffle time by 9%.

5.3.2 Multiple Paths

As mentioned earlier, emerging data center topologies aim to provide multiple paths between each source and destination pair. In this experiment, we evaluate whether multiple-path aware protocols can achieve higher throughput. We use MPTCP[8] [87] as a representative protocol. In this experiment we benchmark the performance of our sort application using TCP, TCP with bonding, and MPTCP. Out setup consists of 9 containers: 1 master and 8 workers. We vary the input dataset size between 1 GB and 4GB[9]. Figure 5.8 plots the shuffle time for TCP, TCP with bonding, and MPTCP. Surprisingly, MPTCP performs poorly in comparison to standard TCP. We attribute this to 3 factors: (1) Additional checksum for data sequence mapping (DSM) [87],

[8] `http://mptcp.info.ucl.ac.be/`

[9] Due to a bug in the MPTCP kernel implementation we are unable to scale out beyond 9 hosts.

(2) The use of short-flows for which MPTCP achieves lower throughput [21], and (3) The immaturity of the MPTCP kernel implementation. Figure 5.9 shows the CPU and network utilization of TCP and MPTCP for a 4 GB dataset. MPTCP does not ramp up to achieve the same throughput as TCP. We believe that these results do not accurately reflect the true potential of MPTCP and evaluation on a different set-up would yield more representative results.

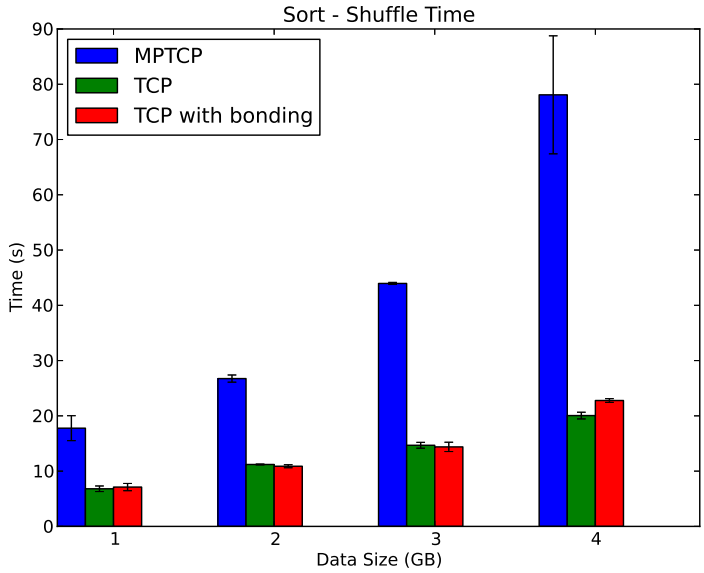
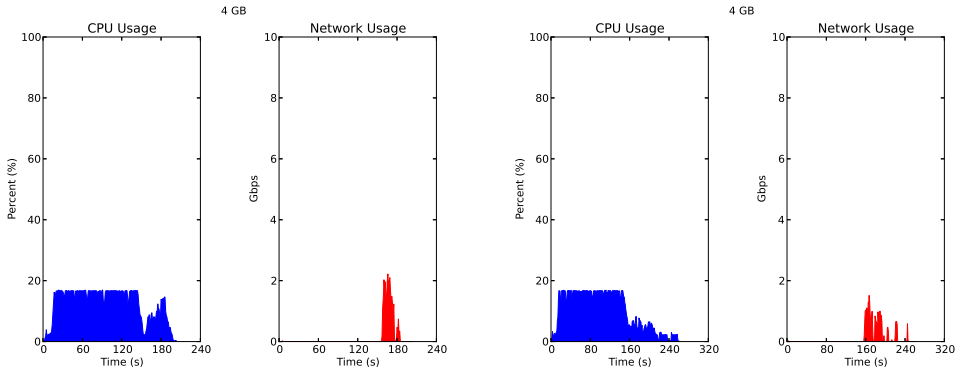


Figure 5.8: Sort TCP vs. MPTCP - Shuffle Time - Star topology

5.3.3 Delay-based vs. AQM-based congestion control

We now investigate whether active queue management (AQM)-based congestion control can outperform the standard delay-based one. In the former, RTT measurements are used as an indication of congestion while in the latter, explicit feedback from the switches is employed. We use data center TCP (DCTCP)[10] [14] as a representative protocol for AQM. Figure 5.10 plots the performance of standard delay-based TCP against AQM-based DCTCP on

[10] `https://github.com/myasuda/DCTCP-Linux`



(a) TCP (b) MPTCP

Figure 5.9: CPU and Network usage of sort using MPTCP for a 4 GB dataset

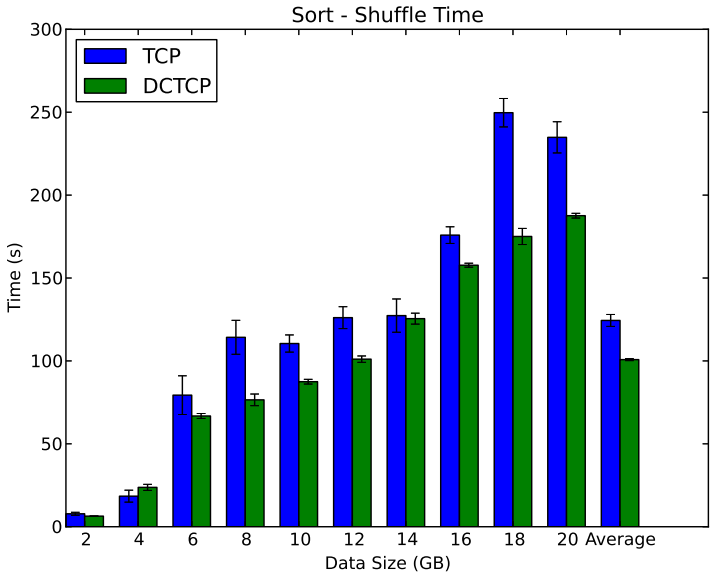


Figure 5.10: Sort TCP vs. DCTCP - Shuffle Time - Star topology

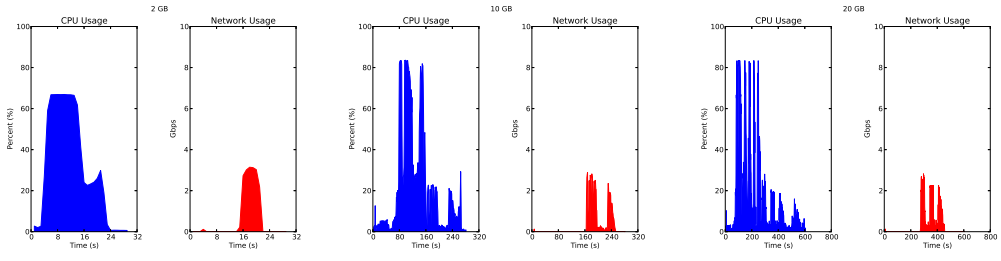
our sort application. DCTCP reduces the shuffle time by 20% on average as it achieves higher throughput.

Figure 5.11 shows the CPU and network utilization under DCTCP. In contrast to Figure 5.3 we see that network traffic spikes are closely packed which results in higher throughput. This can be attributed to DCTCP's ability to trade convergence for higher throughput. To achieve this, DCTCP sources

Data Size (GB)	DCTCP	TCP
2	0.799	0.793
4	0.902	0.904
6	0.801	0.943
8	0.759	0.964
10	0.896	0.938
12	0.910	0.945
14	0.924	0.968
16	0.940	0.965
18	0.958	0.969
20	0.944	0.955
Average	0.883	0.934

Table 5.1: Jain's fairness index for DCTCP and TCP

incrementally adjust their window size based on the extent of congestion. In contrast, TCP throughput has more variation At the same time we note that TCP ensures more fairness for individual flows. Table 5.1 shows the Jain's fairness index [88] for both TCP and DCTCP. TCP's fairness is on average 0.05 higher than DCTCP. For shorter flows DCTCP does not converge quickly, as a result it does not achieve fair share equilibrium. For larger flows the fairness index of DCTCP matches that of TCP due to adequate time to converge to equilibrium.



(a) 2 GB - 32 maps (b) 10 GB - 160 maps (c) 20 GB - 320 maps

Figure 5.11: CPU and Network usage of sort using DCTCP

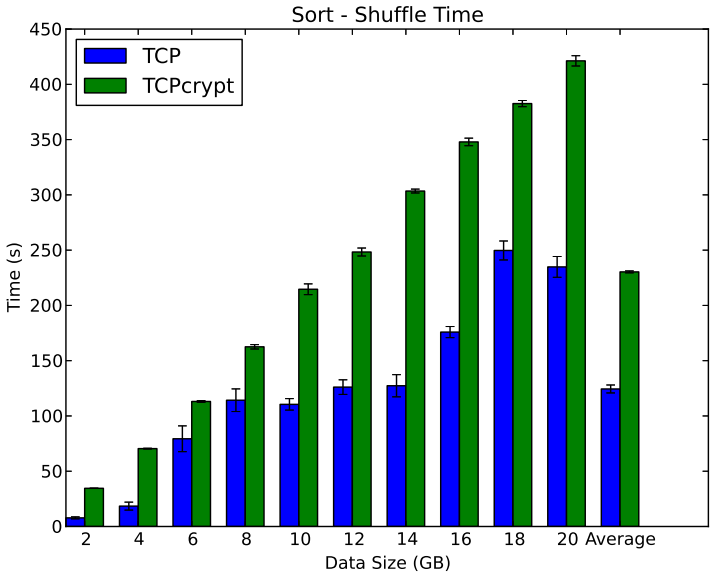


Figure 5.12: Sort TCP vs. TCPcrypt - Shuffle Time - Star topology

5.3.4 Transport Encryption

In cases where the same framework is being used by multiple users without any VLAN isolation, users might want to protect their data by enforcing encryption. This can be done at the application layer but that would result in a very high overhead. We investigate whether transport layer encryption can be applied with an acceptable overhead. We use TCPcrypt[11] as a representative protocol and benchmark it against standard TCP. Figure 5.12 shows the shuffle time of both protocols. On average, TCPcrypt results in an 85% increase in data transfer time. It is worth noting that we used the user-space implementation of TCPcrypt. A kernel-space implementation would potentially decrease the overhead.

[11] `https://github.com/sorbo/tcpcrypt`

Receivers	Input size (GB)	Maps	Reduces
10	2.5	40	10
20	5	80	20
30	7.5	120	30
40	10	160	40

Table 5.2: Parameters for concurrent flows experiment

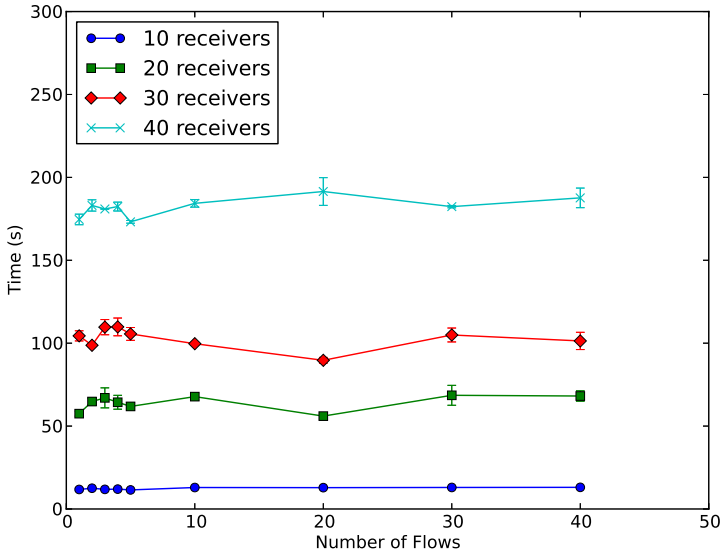


Figure 5.13: Shuffle time with increasing number of concurrent flows

5.3.5 Number of concurrent flows

In this experiment, we check the effect of varying the number of concurrent connections at each worker to fetch its input during the shuffle phase. In addition, we also study whether the number of receivers has any bearing on the shuffle time. We vary the number of map and reduce tasks and the input data size for our sort application (given in Table 5.2) to force each reduce task to fetch approximately 256 MB of data. Figure 5.13 shows shuffle time for increasing number of receivers as a function of the number of flows. Surprisingly, the number of concurrent flows has no effect on the shuffle time. To find out whether this is due to the nature of the application or our ex-

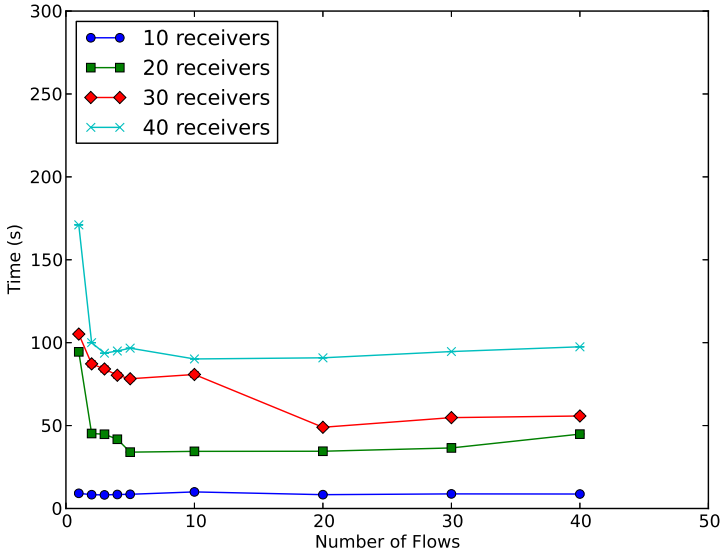


Figure 5.14: Shuffle time with increasing number of concurrent flows using jumbo Ethernet frames

perimental setup, we use `iperf`[12] to measure the throughput of concurrent senders . For even a small number of senders as 16, the average throughput is 140.937 Mbps, which is far below the theoretical value of 1 Gbps. Increasing the maximum transmission unit (MTU)[13] of Ethernet to 9000 byte jumbo frames [89] substantially increases the per sender/receiver pair throughput to 881.5 Mbps. Having achieved full-bisection bandwidth between sender/receiver pairs, we repeat the original experiment on our improved setup and plot the result in Figure 5.14. The results point to 3 main takeaways, (1) The effect of concurrent flows comes into play when the receivers have to fetch more than one file from each sender, (2) Even having two concurrent connections substantially improves performance, (3) Beyond a certain point we encounter diminishing returns. In fact, due to the three point, systems such as Hadoop limit the number of concurrent flows per receiver to 5[14].

[12] `http://sourceforge.net/projects/iperf/`

[13] MTU is maximum unit of data in bytes that is passed by any layer of the network stack.

[14] This limit is also set to keep the number of threads in check and also to mitigate incast [37].

This analysis proves that using the number of flows to implement weighted transport scheduling can effect the shuffle time.

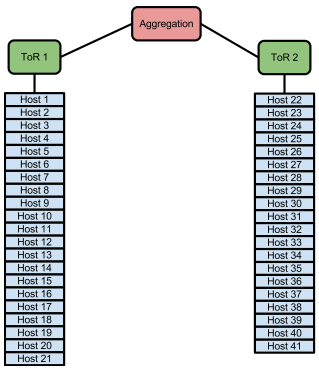


Figure 5.15: 2 rack topology

5.4 Performance of CIEL under Mission Control

In this section, we evaluate the performance of CIEL under Mission Control for a 2 rack topology and 3 switches. Rack 1 has 21 hosts (containers) while Rack 2 has 20 hosts (containers). Hosts within each rack are connected via a 1 Gbps switch each while the racks are connected via another 1 Gbps. The topology is shown in Figure 5.15. The master is run on a host in Rack 1 while the rest of the hosts in the two racks run workers. We evaluate the performance of our sort application for different input sizes. We first do the evaluation using the default TCP Flight Controller. We then repeat the experiment by turning on ECN in all switches. This prompts Mission Controller to switch to the DCTCP Flight Controller. Figure 5.16 plots the shuffle time for this experiment. On average, DCTCP results in a 9% improvement in shuffle time. DCTCP aims to keep switch buffer occupancies low by reacting to the extent of congestion. By default, for a 1 Gbps network the queue length is limited to 20 packets or 30 KB. To corroborate this, we log the TCP receive queue length of every worker every 125 ms for both TCP

and DCTCP for an input data set of size 10 GB. Figure 5.17 plots the time series of queue length for a single worker. We see that DCTCP is able to keep the queue length within 30 KB while the TCP receive queue is longer and has more variation. As a result, DCTCP mitigates buffer pressure and does not allow a single flow to overrun the shared buffer in the switch.

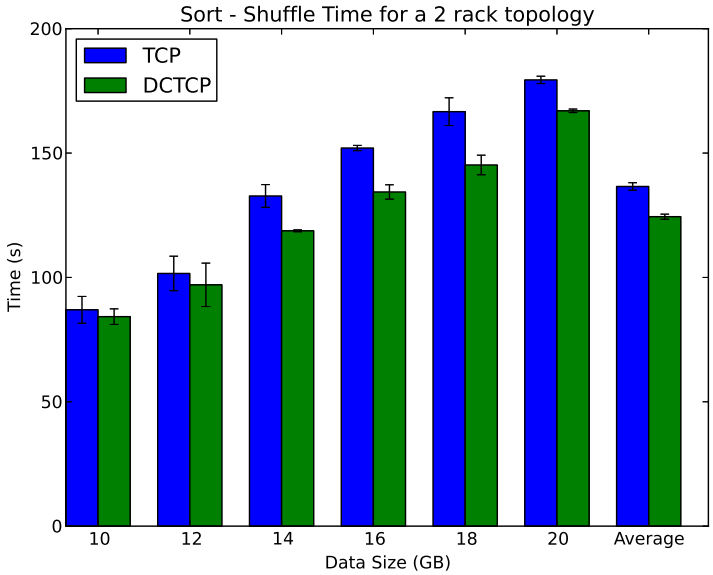
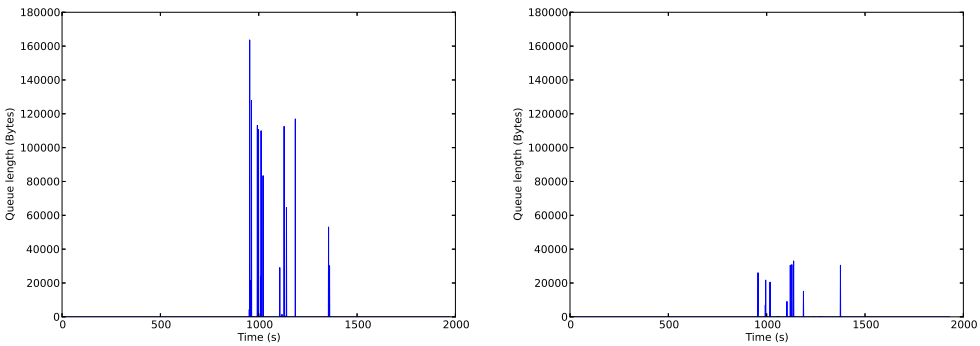


Figure 5.16: Shuffle time for TCP and DCTCP on a two rack topology



(a) TCP (b) DCTCP

Figure 5.17: Time series of receive queue length for TCP and DCTCP

5.5 Performance of Virtual Synchrony primitives

- Multicast time

- Overhead of Isis2

- IronPython vs. cPython

Chapter 6

Summary and Conclusions

Future work:

- Recent work [90] has shown that I/O throughput and latency can vary significantly (order of magnitude) due to the underlying IPC mechanism, the physical and virtual architecture, and OS primitives. Mission Control can be run on top of the proposed FABLE stack to switch to shared memory primitives if the destination is local.
- Mission Control can be merged with Quincy [63] and Mesos [91] to enable data center/cluster-wide CPU, memory and I/O isolation and scheduling.
- It would interesting to see how Mission Control can be used to leverage the transparency of the software stack in Akaros [92] to improve I/O performance.
- 80% of the traffic in a cluster stays within a rack [7]. This information can be used by Mission Control to assign more flows to in-rack transfers.
- In this thesis we only considered an environment under the operational control of a single domain. It is not clear how different transport protocols would interact with each other within the same data center. For instance, DCTCP flows might not ensure fairness to TCP flows [14].

- Our focus in this discussion was on batch-processing systems which require high throughput. In addition to the systems, data centers also run stream processing systems, such as S4 [93] and Percolator [94], which require low latency. The choice of underlying transport semantics is completely different and Mission Control would need to be made aware of these requirements.
- The last point can be generalized to a trade-off between bandwidth and latency. Protocols such as HULL [75] make this trade-off possible. Mission Control can be extended to use global knowledge to optimize this trade-off.
- Mission Control currently supports 4 different transport protocols but the framework is flexible enough to be extended to make use of more protocols such as Deadline-Driven Delivery (D^3) [22], Structured Stream Transport (SST) [76], and ICTCP [23].

Bibliography

[1] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[3] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[4] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[5] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[6] Derek G. Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy, and Steven Hand. CIEL: a universal execution engine for distributed data-flow computing. In *Proceedings of*

the 8th USENIX conference on Networked systems design and implementation, NSDI '11, pages 9–9, Berkeley, CA, USA, 2011. USENIX Association.

[7] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.

[8] R.H. Katz. Tech titans building boom. *Spectrum, IEEE*, 46(2):40 –54, February 2009.

[9] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23:22–28, March 2003.

[10] Daniel Halperin, Srikanth Kandula, Jitendra Padhye, Paramvir Bahl, and David Wetherall. Augmenting data center networks with multi-gigabit wireless links. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 38–49, New York, NY, USA, 2011. ACM.

[11] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T.S. Eugene Ng, Michael Kozuch, and Michael Ryan. c-Through: part-time optics in data centers. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 327–338, New York, NY, USA, 2010. ACM.

[12] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 339–350, New York, NY, USA, 2010. ACM.

[13] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[14] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[15] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 75–86, New York, NY, USA, 2008. ACM.

[16] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. BCube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 63–74, New York, NY, USA, 2009. ACM.

[17] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[18] Jayaram Mudigonda, Praveen Yalagandula, Mohammad Al-Fares, and Jeffrey C. Mogul. SPAIN: COTS data-center Ethernet for multipathing over arbitrary topologies. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI '10, pages 18–18, Berkeley, CA, USA, 2010. USENIX Association.

[19] Bhanu Chandra Vattikonda, George Porter, Amin Vahdat, and Alex C. Snoeren. Practical TDMA for datacenter ethernet. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 225–238, New York, NY, USA, 2012. ACM.

[20] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Brian Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 303–314, New York, NY, USA, 2009. ACM.

[21] Costin Raiciu, Christopher Pluntke, Sebastien Barre, Adam Greenhalgh, Damon Wischik, and Mark Handley. Data center networking with multipath TCP. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets '10, pages 10:1–10:6, New York, NY, USA, 2010. ACM.

[22] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: meeting deadlines in datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 50–61, New York, NY, USA, 2011. ACM.

[23] Haitao Wu, Zhenqian Feng, Chuanxiong Guo, and Yongguang Zhang. ICTCP: Incast Congestion Control for TCP in data center networks. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 13:1–13:12, New York, NY, USA, 2010. ACM.

[24] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: networking data centers, randomly. In *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, HotCloud '11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

[25] Jayaram Mudigonda, Praveen Yalagandula, and Jeffrey C. Mogul. Taming the flying cable monster: a topology design and optimization framework for data-center networks. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIX ATC '11, pages 8–8, Berkeley, CA, USA, 2011. USENIX Association.

[26] Kai Chen, Chuanxiong Guo, Haitao Wu, Jing Yuan, Zhenqian Feng, Yan Chen, Songwu Lu, and Wenfei Wu. Generic and automatic address configuration for data center networks. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 39–50, New York, NY, USA, 2010. ACM.

[27] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 39–50, New York, NY, USA, 2009. ACM.

[28] Jayaram Mudigonda, Praveen Yalagandula, Jeff Mogul, Bryan Stiekes, and Yanick Pouffary. NetLord: a scalable multi-tenant network architecture for virtualized datacenters. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 62–73, New York, NY, USA, 2011. ACM.

[29] Chuanxiong Guo, Guohan Lu, Helen J. Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. SecondNet: a data center network virtualization architecture with bandwidth guaran-

tees. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 15:1–15:12, New York, NY, USA, 2010. ACM.

[30] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 242–253, New York, NY, USA, 2011. ACM.

[31] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI '11, pages 23–23, Berkeley, CA, USA, 2011. USENIX Association.

[32] Henrique Rodrigues, Jose Renato Santos, Yoshio Turner, Paolo Soares, and Dorgival Guedes. Gatekeeper: supporting bandwidth guarantees for multi-tenant datacenter networks. In *Proceedings of the 3rd conference on I/O virtualization*, WIOV '11, pages 6–6, Berkeley, CA, USA, 2011. USENIX Association.

[33] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI '10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[34] Saamer Akhshabi and Constantine Dovrolis. The evolution of layered protocol stacks leads to an hourglass-shaped architecture. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 206–217, New York, NY, USA, 2011. ACM.

[35] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. Understanding TCP incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, WREN '09, pages 73–82, New York, NY, USA, 2009. ACM.

[36] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 202–208, New York, NY, USA, 2009. ACM.

[37] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I. Jordan, and Ion Stoica. Managing data transfers in computer clusters with Orchestra.

In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 98–109, New York, NY, USA, 2011. ACM.

[38] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving MapReduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI '08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.

[39] Srikanth Kandula, Jitendra Padhye, and Paramvir Bahl. Flyways to de-congest data center networks. In *Proceedings of the Eighth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets '09, 2009.

[40] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. The case for fine-grained traffic engineering in data centers. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, INM/WREN '10, pages 2–2, Berkeley, CA, USA, 2010. USENIX Association.

[41] Sahan Gamage, Ardalan Kangarlou, Ramana Rao Kompella, and Dongyan Xu. Opportunistic flooding to improve TCP transmit performance in virtualized clouds. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 24:1–24:14, New York, NY, USA, 2011. ACM.

[42] Ardalan Kangarlou, Sahan Gamage, Ramana Rao Kompella, and Dongyan Xu. vSnoop: Improving TCP Throughput in Virtualized Environments via Acknowledgement Offload. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[43] Guohui Wang and T. S. Eugene Ng. The impact of virtualization on network performance of amazon EC2 data center. In *Proceedings of the 29th conference on Information communications*, INFOCOM '10, pages 1163–1171, Piscataway, NJ, USA, 2010. IEEE Press.

[44] J. Rothschild. High performance at massive scale - lessons learned at Facebook, October 2009.

[45] Tom Lyon. TCP issues in the Data Center. In *The Future of TCP: Train-wreck or Evolution*, Stanford, CA, 2008.

[46] D. Clark. The design philosophy of the DARPA Internet protocols. In *Conference proceedings on communications, architectures & protocols*, SIGCOMM '88, pages 106–114, New York, NY, USA, 1988. ACM.

[47] T.R. Henderson and R.H. Katz. Transport protocols for Internet-compatible satellite networks. *Selected Areas in Communications, IEEE Journal on*, 17(2):326 –344, feb 1999.

[48] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. MicroTE: fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 8:1–8:12, New York, NY, USA, 2011. ACM.

[49] Hussam Abu-Libdeh, Paolo Costa, Antony Rowstron, Greg O'Shea, and Austin Donnelly. Symbiotic routing in future data centers. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 51–62, New York, NY, USA, 2010. ACM.

[50] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI '10, pages 17–17, Berkeley, CA, USA, 2010. USENIX Association.

[51] Andrea Bittau, Michael Hamburg, Mark Handley, David Mazires, and Dan Boneh. The Case for Ubiquitous Transport-Level Encryption. In *USENIX Security Symposium '10*, pages 403–418, 2010.

[52] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Computer Communication Review*, 39(1):68–73, December 2008.

[53] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica, and et al. Above the Clouds: A Berkeley View of Cloud Computing. *EECS Department University of California Berkeley Tech Rep UCBEECS200928*, (UCB/EECS-2009-28):25, 2009.

[54] Apache hadoop. `http://hadoop.apache.org/`.

[55] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D. Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference*

on Management of data, SIGMOD '07, pages 1029–1040, New York, NY, USA, 2007. ACM.

[56] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. HaLoop: efficient iterative data processing on large clusters. *Proc. VLDB Endow.*, 3(1-2):285–296, September 2010.

[57] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud '10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[58] Derek G. Murray and Steven Hand. Scripting the cloud with skywriting. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud '10, pages 12–12, Berkeley, CA, USA, 2010. USENIX Association.

[59] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig Latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1099–1110, New York, NY, USA, 2008. ACM.

[60] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming*, 13(4):277–298, October 2005.

[61] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. DryadLINQ: a system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI '08, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.

[62] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 265–278, New York, NY, USA, 2010. ACM.

[63] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd sympo-*

sium on Operating systems principles, SOSP '09, pages 261–276, New York, NY, USA, 2009. ACM.

[64] Eric A. Brewer. Towards robust distributed systems (abstract). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, PODC '00, pages 7–, New York, NY, USA, 2000. ACM.

[65] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[66] Werner Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, January 2009.

[67] Dan Pritchett. BASE: An ACID Alternative. *ACM Queue*, 6(3):48–55, May 2008.

[68] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the eleventh ACM Symposium on Operating Systems principles*, SOSP '87, pages 123–138, New York, NY, USA, 1987. ACM.

[69] Ken Birman. A history of the virtual synchrony replicationmodel. In Bernadette Charron-Bost, Fernando Pedone, and Andr Schiper, editors, *Replication*, volume 5959 of *Lecture Notes in Computer Science*, pages 91–120. Springer Berlin Heidelberg, 2010.

[70] Kashi Venkatesh Vishwanath, Albert Greenberg, and Daniel A. Reed. Modular data centers: how to design them? In *Proceedings of the 1st ACM workshop on Large-Scale system and application performance*, LSAP '09, pages 3–10, New York, NY, USA, 2009. ACM.

[71] Meg Walraed-Sullivan, Radhika Niranjan Mysore, Malveeka Tewari, Ying Zhang, Keith Marzullo, and Amin Vahdat. ALIAS: scalable, decentralized label assignment for data centers. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 6:1–6:14, New York, NY, USA, 2011. ACM.

[72] Paolo Costa, Austin Donnelly, Antony Rowstron, and Greg O'Shea. Camdoop: Exploiting In-Network Aggregation for Big Data Applications. In *Proceedings of the 9^{th} USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*, San Jose, CA, US, April 2012. Usenix.

[73] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath TCP. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 266–277, New York, NY, USA, 2011. ACM.

[74] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. DevoFlow: scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 254–265, New York, NY, USA, 2011. ACM.

[75] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is More: Trading a little Bandwidth for Ultra-Low Latency in the Data Center. In *Proceedings of the 9^{th} USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*, San Jose, CA, US, April 2012. USENIX.

[76] Bryan Ford. Structured streams: a new transport abstraction. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 361–372, New York, NY, USA, 2007. ACM.

[77] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, WREN '09, pages 65–72, New York, NY, USA, 2009. ACM.

[78] Yuan Yu, Pradeep Kumar Gunda, and Michael Isard. Distributed aggregation for data-parallel computing: interfaces and implementations. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 247–260, New York, NY, USA, 2009. ACM.

[79] Jiaxing Zhang, Hucheng Zhou, Rishan Chen, Xuepeng Fan, Zhenyu Guo, Haoxiang Lin, Jack Y. Li, Wei Lin, Jingren Zhou, and Lidong Zhou. Optimizing Data Shufing in Data-Parallel Computation by Understanding User-Defined Functions. In *Proceedings of the 9^{th} USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*, San Jose, CA, US, April 2012. USENIX.

[80] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility.

In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*, San Jose, CA, US, April 2012. USENIX.

[81] Alan Shieh, Srikanth Kandula, Albert Greenberg, and Changhoon Kim. Seawall: performance isolation for cloud datacenter networks. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud '10, pages 1–1, Berkeley, CA, USA, 2010. USENIX Association.

[82] Gautam Kumar, Mosharaf Chowdhury, Sylvia Ratnasamy, and Ion Stoica. A case for performance-centric network allocation. In *Proceedings of the 4th USENIX conference on Hot topics in cloud computing*, HotCloud '12, Berkeley, CA, USA, 2012. USENIX Association.

[83] D. G. Murray. *A distributed execution engine supporting data-dependent control flow*. PhD thesis, University of Cambridge, 2011.

[84] Kenneth P. Birman, Daniel A. Freedman, Qi Huang, and Patrick Dowell. Overcoming CAP with Consistent Soft-State Replication. *Computer*, 45:50–58, 2012.

[85] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[86] Ben Pfaff, Justin Pettit, Teemu Koponen, Keith Amidon, Martin Casado, and Scott Shenker. Extending networking into the virtualization layer. In *In 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII)*, New York City, NY, October 2009.

[87] Costin Raiciu, Christoph Paasch, Sébastien Barré, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*, San Jose, CA, US, April 2012. USENIX.

[88] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. Technical report, Digital Equipment Corporation, September 1984.

[89] Jeff Chase, Andrew Gallatin, and Ken Yocum. End-System Optimizations for High-Speed TCP. *IEEE Communications Magazine*, 39:68–74, 2000.

[90] Steven Smith, Anil Madhavapeddy, Christopher Smowton, Malte Schwarzkopf, Richard Mortier, Robert M. Watson, and Steven Hand. The Case for Reconfigurable I/O Channels. In *Runtime Environments, Systems, Layering and Virtualized Environments (RESoLVE '12)*, Workshop at ASPLOS '12, London, UK, 2012. ACM.

[91] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: a platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI '11, pages 22–22, Berkeley, CA, USA, 2011. USENIX Association.

[92] Barret Rhoden, Kevin Klues, David Zhu, and Eric Brewer. Improving per-node efficiency in the datacenter with new OS abstractions. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 25:1–25:8, New York, NY, USA, 2011. ACM.

[93] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed stream computing platform. In *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops*, ICDMW '10, pages 170–177, Washington, DC, USA, 2010. IEEE Computer Society.

[94] Daniel Peng and Frank Dabek. Large-scale incremental processing using distributed transactions and notifications. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–15, Berkeley, CA, USA, 2010. USENIX Association.