Research in (Distributed, Operating and Networked) Systems

Alysson Bessani
LaSIGE
Academia

- Highly competitive environment
  - Impact
  - Funding
  - Publishing

- Researchers are high-competition athletes
Systems Main Topics

• **Operating Systems** (VM Hypervisors, multi-core kernels, fast IO, use of new hardware)
• **Distributed Systems** (replication, scalability, security, test and debugging)
• **Cloud computing** (big data, schedulers)
• **Storage** (database, file system)
• **Field data analysis** (faultloads, workloads)
• **Remarkable/novel systems in production** (Facebook, Google, Amazon, Microsoft)
Xen and the Art of Virtualization

Paul Barham†, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer†, Ian Pratt, Andrew Warfield

University of Cambridge Computer Laboratory
15 JJ Thomson Avenue, Cambridge, UK, CB3 0FD
{firstname.lastname}@cl.cam.ac.uk

ABSTRACT

Numerous systems have been designed which use virtualization to subdivide the ample resources of a modern computer. Some require specialized hardware, or cannot support commodity operating systems. Some target 100% binary compatibility at the expense of performance. Others sacrifice security or functionality for speed. Few offer resource isolation or performance guarantees; most provide only best-effort provisioning, risking denial of service.

This paper presents Xen, an x86 virtual machine monitor which allows multiple commodity operating systems to share conventional hardware in a safe and resource managed fashion, but without sacrificing either performance or functionality. This is achieved by providing an idealized virtual machine abstraction to which operating systems such as Linux, BSD and Windows XP, can be ported with minimal effort.

Our design is targeted at hosting up to 100 virtual machine instances simultaneously on a modern server. The virtualization approach taken by Xen is extremely efficient: we allow operating systems such as Linux and Windows XP to be hosted simultaneously for a negligible performance overhead — at most a few percent compared with the unvirtualized case. We considerably outperform competing commercial and freely available solutions in a range of microbenchmarks and system-wide tests.

1. INTRODUCTION

Modern computers are sufficiently powerful to use virtualization to present the illusion of many smaller virtual machines (VMs), each running a separate operating system instance. This has led to a resurgence of interest in VM technology. In this paper we present Xen, a high performance resource-managed virtual machine monitor (VMM) which enables applications such as server consolidation [42, 8], co-located hosting facilities [14], distributed web services [43], secure computing platforms [12, 16] and application mobility [26, 37].

Successful partitioning of a machine to support the concurrent execution of multiple operating systems poses several challenges. Firstly, virtual machines must be isolated from one another: it is not acceptable for the execution of one to adversely affect the performance of another. This is particularly true when virtual machines are owned by mutually untrusting users. Secondly, it is necessary to support a variety of different operating systems to accommodate the heterogeneity of popular applications. Thirdly, the performance overhead introduced by virtualization should be small.

Xen hosts commodity operating systems, albeit with some source modifications. The prototype described and evaluated in this paper can support multiple concurrent instances of our XenoLinux guest operating system; each instance exports an application binary interface identical to a non-virtualized Linux 2.4. Our port of Windows
MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program’s execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google’s clusters every day.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the map and reduce primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a map operation to each logical “record” in our input in order to compute a set of intermediate key/value pairs, and then applying a reduce operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with userspecified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.

The major contributions of this work are a simple and powerful interface that enables automatic parallelization...
Zyzzyva: Speculative Byzantine Fault Tolerance

Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong
Dept. of Computer Sciences
University of Texas at Austin
{kotla,lorenzo,dahlin,aclement,elwong}@cs.utexas.edu

ABSTRACT
We present Zyzzyva, a protocol that uses speculation to reduce the cost and simplify the design of Byzantine fault tolerant state machine replication. In Zyzzyva, replicas respond to a client’s request without first running an expensive three-phase commit protocol to reach agreement on the order in which the request must be processed. Instead, they optimistically adopt the order proposed by the primary and respond immediately to the client. Replicas can thus become temporarily inconsistent with one another, but clients detect inconsistencies, help correct replicas converge on a single total ordering of requests, and only rely on responses that are consistent with this total order. This approach allows Zyzzyva to reduce replication overheads to near their theoretical minima.

non-fail-stop behavior in real systems [2, 5, 6, 27, 30, 32, 36, 39, 40] suggest that BFT may yield significant benefits even without resorting to n-version programming [4, 15, 33]. Third, improvements to the state of the art in BFT replication techniques [3, 9, 10, 18, 33, 41] make BFT replication increasingly practical by narrowing the gap between BFT replication costs and costs already being paid for non-BFT replication. For example, by default, the Google file system uses 3-way replication of storage, which is roughly the cost of BFT replication for $f = 1$ failures with 4 agreement nodes and 3 execution nodes [41].

This paper presents Zyzzyva\(^1\), a new protocol that uses speculation to reduce the cost and simplify the design of BFT state machine replication [19, 35]. Like traditional state machine replication protocols [9, 33, 41], a primary proposes an order on client requests to the other replicas. In Zyzzyva,
Flexible, Wide-Area Storage for Distributed Systems with WheelFS

Jeremy Stribling, Yair Sovran† Irene Zhang, Xavid Pretzer, Jinyang Li‡ M. Frans Kaashoek, and Robert Morris
MIT CSAIL †New York University

Abstract

WheelFS is a wide-area distributed storage system intended to help multi-site applications share data and gain fault tolerance. WheelFS takes the form of a distributed file system with a familiar POSIX interface. Its design allows applications to adjust the tradeoff between prompt visibility of updates from other sites and the ability for sites to operate independently despite failures and long delays. WheelFS allows these adjustments via semantic cues, which provide application control over consistency, failure handling, and file and replica placement.

WheelFS is implemented as a user-level file system and is deployed on PlanetLab and Emulab. Three applications (a distributed Web cache, an email service and large file distribution) demonstrate that WheelFS’s file system interface simplifies construction of distributed applications by allowing reuse of existing software. These applications would perform poorly with the strict semantics implied by a traditional file system interface, but by providing cues to WheelFS they are able to achieve good performance. Measurements show that applications built on WheelFS deliver comparable performance to services such as CoralCDN and BitTorrent that use specialized wide-area storage systems.

model affects the sharing/independence tradeoff: stronger forms of consistency usually involve servers or quorums of servers that serialize all storage operations, whose unreliability may force delays at other sites [23]. The storage system’s data and meta-data placement decisions also affect site independence, since data placed at a distant site may be slow to fetch or unavailable.

The wide-area file system introduced in this paper, WheelFS, allows application control over the sharing/independence tradeoff, including consistency, failure handling, and replica placement. Each application can choose a tradeoff between performance and consistency, in the style of PRACTI [8] and PADs [9], but in the context of a file system with a POSIX interface.

Central decisions in the design of WheelFS including defining the default behavior, choosing which behaviors applications can control, and finding a simple way for applications to specify those behaviors. By default, WheelFS provides standard file system semantics (close-to-open consistency) and is implemented similarly to previous wide-area file systems (e.g., every file or directory has a primary storage node). Applications can adjust the default semantics and policies with semantic cues. The set of cues is small (around 10) and directly addresses the main challenges of wide-area networks (orders of magni-
Abstract

We present the first large-scale analysis of hardware failure rates on a million consumer PCs. We find that many failures are neither transient nor independent. Instead, a large portion of hardware-induced failures are recurrent: a machine that crashes from a fault in hardware is up to two orders of magnitude more likely to crash a second time. For example, machines with at least 30 days of accumulated CPU time over an 8 month period had a 1 in 190 chance of crashing due to a CPU subsystem fault. Further, machines that crashed once had a probability of 1 in 3.3 of crashing a second time. Our study examines failures due to faults within the CPU, DRAM, and disk subsystems. Our analysis spans desktops and laptops, CPU vendor, overclocking, underclocking, generic vs. brand name, and characteristics such as machine speed and calendar age. Among our many results, we find that CPU fault rates are correlated with the number of cycles executed, underclocked machines are significantly more reliable than machines running at their rated speed, and laptops are more reliable than desktops.
Dynamo: Amazon’s Highly Available Key-value Store
Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels
Amazon.com

ABSTRACT
Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many data centers around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon’s core services use to provide an “always-on” experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

One of the lessons our organization has learned from operating Amazon’s platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon’s software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.
Abstract

We introduce a simple data model and API tailored for serving the social graph, and TAO, an implementation of this model. TAO is a geographically distributed data store that provides efficient and timely access to the social graph for Facebook’s demanding workload using a fixed set of queries. It is deployed at Facebook, replacing memcache for many data types that fit its model. The system runs on thousands of machines, is widely distributed, and provides access to many petabytes of data. TAO can process a billion reads and millions of writes each second.

1 Introduction

Facebook has more than a billion active users who record their relationships, share their interests, upload text, images, and video, and curate semantic information about their data [2]. The personalized experience of social applications comes from timely, efficient, and scalable access to this flood of data, the social graph. In this paper we introduce TAO, a read-optimized graph data store we have built to handle a demanding Facebook workload.
iSeeYou: Disabling the MacBook Webcam Indicator LED

Matthew Brocker  
*Johns Hopkins University*

Stephen Checkoway  
*Johns Hopkins University*

**Abstract**

The ubiquitous webcam indicator LED is an important privacy feature which provides a visual cue that the camera is turned on. We describe how to disable the LED on a class of Apple internal iSight webcams used in some versions of MacBook laptops and iMac desktops. This enables video to be captured without any visual indication to the user and can be accomplished entirely in user space by an unprivileged (non-root) application.

The same technique that allows us to disable the LED, namely reprogramming the firmware that runs on the iSight, enables a virtual machine escape whereby malware running inside a virtual machine reprograms the camera to act as a USB Human Interface Device (HID) keyboard which executes code in the host operating system.

We build two proofs-of-concept: (1) an OS X application, *iSeeYou*, which demonstrates capturing video with the LED disabled; and (2) a virtual machine escape that launches *Terminal.app* and runs shell commands. To defend against these and related threats, we build an OS X kernel extension, *iSightDefender*, which prohibits the modification of the iSight’s firmware from user space.

![Image sensor (front)](image1)

![Image sensor (back)](image2)

![Main board (front)](image3)

![Main board (back)](image4)

*Figure 1:* The iSight from a 2008 MacBook we studied.

The value of video evidence is so high that The Washington Post recently reported that the US Federal Bureau of Investigation (FBI), has developed surveillance mal-
 TensorFlow: A system for large-scale machine learning

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng

Google Brain

Abstract

TensorFlow is a machine learning system that operates at large scale and in heterogeneous environments. TensorFlow uses dataflow graphs to represent computation, shared state, and the operations that mutate that state. It maps the nodes of a dataflow graph across many machines in a cluster, and within a machine across multiple computational devices, including multicore CPUs, general-purpose GPUs, and custom-designed ASICs known as Tensor Processing Units (TPUs). This architecture gives flexibility to the application developer: whereas in previous “parameter server” designs the management of shared state is built into the system, TensorFlow enables developers to experiment with novel optimizations and training algorithms. TensorFlow supports a variety of applications, with a focus on training and inference on deep neural networks. Several Google services use TensorFlow in production, we have released it as an open-source project, and it has become widely used for machine learning research. In this paper, we describe the TensorFlow dataflow model and demonstrate the compelling performance that TensorFlow achieves for several real-world applications.

1 Introduction

In recent years, machine learning has driven advances in many different fields [3, 5, 24, 25, 29, 31, 42, 47, 50, 52, 57, 67, 68, 72, 76]. We attribute this success to the invention of more sophisticated machine learning models [44, 54], the availability of large datasets for tackling problems in these fields [9, 64], and the development of software platforms that enable the easy use of large amounts of computational resources for training such models on these large datasets [14, 20].

We have developed the TensorFlow system for experimenting with new models, training them on large datasets, and moving them into production. We have based TensorFlow on many years of experience with our first-generation system, DistBelief [20], both simplifying and generalizing it to enable researchers to explore a wider variety of ideas with relative ease. TensorFlow supports both large-scale training and inference: it efficiently uses hundreds of powerful (GPU-enabled) servers for fast training, and it runs trained models for inference in production on various platforms, ranging from large distributed clusters in a datacenter, down to running locally on mobile devices. At the same time, it is flexible enough to support experimentation and research into new machine learning models and system-level optimizations.

TensorFlow uses a unified dataflow graph to represent both the computation in an algorithm and the state on which the algorithm operates. We have drawn inspiration from the high-level programming models of dataflow systems [12, 21, 34] and the low-level efficiency of parameter...
GINJA: One-dollar Cloud-based Disaster Recovery for Databases

Joel Alcântara, Tiago Oliveira, Alysson Bessani
LaSIGE – Faculdade de Ciências, Universidade de Lisboa
Portugal

Abstract
Disaster Recovery (DR) is a crucial feature to ensure availability and data protection in modern information systems. A common DR approach requires the replication of services in a set of virtual machines running in the cloud as backups. This leads to considerable monetary costs and managing efforts to keep such cloud VMs. We present GINJA, a DR solution for transactional database management systems (DBMS) that uses only cloud storage services such as Amazon S3. GINJA works at file-system level to efficiently capture and replicate data updates to a remote cloud storage service, achieving three important goals: (1) reduces the costs for maintaining a cloud-based DR to less than one dollar per month for relevant databases’ sizes and workloads (up to $222 \times$ less than the traditional approach of having a DBMS replica in a cloud VM); (2) allows a precise control of the operational costs, durability and performance trade-offs; and (3) introduces a small performance overhead to the DBMS (e.g., less than $5\%$ overhead for the TPC-C workload with $\approx 10$ seconds of data loss in case of disasters).

CCS Concepts • Computer systems organization → Dependable and fault-tolerant systems and networks; • Information systems → Data replication tools;

Keywords Disaster recovery, Databases, Cloud

a dedicated infrastructure [49]. System operators can thus rely on cloud providers to host a portion (or even full copies) of their system and, if the primary site goes offline, they can quickly assume the service provision.

Cloud-based disaster recovery mechanisms require different approaches to deal with stateless and stateful services. For the former, administrators only have to store server VM images to enable the services to be started when required. For stateful services such as databases, there are basically two options: periodically storing state snapshots, or maintaining a warm backup on the cloud [34]. The first approach is known as Backup and Restore while the later is sometimes called Pilot Light, in the sense that this backup replica can spark a whole backup infrastructure if needed [41]. The replication protocol for maintaining such replica in the cloud can be implemented at different layers, such as within the service itself [11, 14, 33], in the virtualization platform [40, 50], or at the storage level [31, 39].

Despite all these options, data loss is still a common event with severe consequences. Although statistics about data losses and its effects are sometimes misleading [27], recent surveys showed that data loss costs $1.7$ Trillion per year for medium and big companies [35]. Few years ago a survey by Symantec showed that $40\%$ of Small and Medium Enterprises (SMEs) do not do regular backups [44]. We believe the situation improved in the last years but...
What these papers have in common?

- Lots of authors in most of them
- Big papers (bigger than most journals)
- Practical problems and systems
- Lots of graphs and experimental eval.
- Very good venues (acceptance rate low)
What people like/don’t like?

• **Good**
  – Practical and timely problems
  – Principled solution
  – Open-source implementations*
  – Solid experimental evaluation

• **Bad**
  – Problems of theoretical interest
  – Too complex solutions*
  – Only specific hacks and tricks as a solution
  – Simulated or analytical evaluation
Paper Organization

• Introduction
• Problem Statement
• Solution
• Implementation
• Evaluation
• Related Work
• Conclusion

Should be very strong

Big and based on application workloads
What are the top conferences?

- **Systems**: SOSP/OSDI, EuroSys, USENIX ATC, FAST
- **Distributed Systems**: ICDCS, IPDPS, Middleware
- **Dependability**: DSN, SRDS, ISSRE
- **Theory**: PODC, DISC, OPODIS, SSS
- **Security**: Security & Privacy (Oakland), CCS, USENIX Security, NDSS, Crypto
- **Networks**: SIGCOMM, INFOCOM, NSDI, CoNEXT
- **Real-time**: RTSS, RTAS, EuroMicro

(there are others, of course...)
Doing research in systems…

• Read a lot
• Formalization and abstraction
• Implement and know the systems
• Write something every day
Some References

  - Book about style of written English, recommended by 10 out of 10 computer science professors in the US

- Levin & Redell. *How (and how not) to write a good systems paper*.
  - [http://www.usenix.org/event/samples/submit/advice.html](http://www.usenix.org/event/samples/submit/advice.html)

- Pages of professors Priya Nar. and Mike Dahlin