The future of Computer Science

Dave Cliff, University of Bristol
(and proud Leeds Class of ’87 graduate)
The future of Computer Science

Dave Cliff, University of Bristol
(and proud Leeds Class of ’87 graduate)
Predicting the future: near term easy
Predicting the future: the last 40 yrs
Moore’s Law

Number of transistors doubling every 18 months.

Number of transistors doubling every 24 months.
Predicting the future: the last 40 yrs
Predicting the future: the next 40 yrs
Predicting the future: the next 40 yrs
Stuttering

Transistors per chip, '000
Clock speed (max), MHz
Thermal design power*, w

Chip introduction dates, selected

Transistors bought per $, m

2002 04 06 08 10 12 15

Log scale

10^7

10^5

10^3

10

10^{-1}

1970 75 80 85 90 95 2000 05 10 15

Sources: Intel; press reports; Bob Colwell; Linley Group; IB Consulting; The Economist

*Maximum safe power consumption
Source: HP Labs c.2004
HPL Quantum Nanowires

9 nm
Amazon EC2 Elastic GPUs allow you to easily attach low-cost graphics acceleration to a wide range of EC2 instances over the network. Simply choose an instance with the right amount of compute, memory, and storage for your application, and then use Elastic GPUs to add the GPU resources needed to accelerate the graphics performance of your application for a fraction of the cost of standalone graphics instances.

Amazon EC2 Elastic GPUs make it easy to attach graphics acceleration to existing Amazon EC2 instances in much the same way as attaching Amazon EBS volumes. With Elastic GPUs, you can configure the right amount of graphics acceleration to your particular workload without being constrained by fixed hardware configurations and limited GPU selection. Elastic GPUs support OpenGL 4.0 and offer up to 8GB of GPU memory, making them ideally suited for any workload that needs a small amount of additional GPU such as virtual desktops, gaming, industrial design, or HPC.
The Datacenter as a Computer
An Introduction to the Design of Warehouse-Scale Machines
Second Edition

Luiz André Barroso
Jimmy Clidaras
Urs Hölzle
Figure 1.1: Sketch of the typical elements in warehouse-scale systems: 1U server (left), 7’ rack with Ethernet switch (middle), and diagram of a small cluster with a cluster-level Ethernet switch/router (right).
Figure 1.2: Picture of a row of servers in a Google WSC, 2012.
Figure 1.3: Storage hierarchy of a WSC.
Figure 4.10: CFD model showing recirculation paths and temperature stratification for a rack with under-provisioned airflow.
Figure 4.5: Airflow schematic of an air-economized datacenter (source: Perspectives, James Hamilton’s blog).
The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google

ABSTRACT
We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is constantly accessed.

1. INTRODUCTION
We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds of even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application
MapReduce: Simplified Data Processing on Large Clusters

by Jeffrey Dean and Sanjay Ghemawat

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a map and a reduce function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. Programmers find the system easy to use: more than ten thousand distinct MapReduce programs have been implemented internally at Google over the past four years, and an average of one hundred thousand MapReduce jobs are executed on Google's clusters every day, processing a total of more than twenty petabytes of data per day.

1 Introduction

Prior to our development of MapReduce, the authors and others at Google implemented hundreds of special-purpose computations that process large amounts of new data, such as crawled web pages. We required large, in-memory computations to be written in terms of MapReduce and we observed that the benefits of MapReduce were often lost in our special-purpose computations. As a result, we designed a new abstraction that allows users to express the simple computations we were trying to perform but below the many details of parallelization, fault tolerance, distributed file system, and load balancing. We call this abstraction the MapReduce model.

We have designed a scalable, high-speed, distributed file system that we call the Google File System (GFS). GFS provides a highly available, high-performance file system for Google's clusters of commodity hardware. GFS is available to applications as a single namespace that spans hundreds of machines and provides a consistent view of a single namespace.

2 Programming Model

The computing jobs are run on a large cluster of commodity servers. Each server has a high-speed connection to a single large disk, and the disk holds all the data for the job. The MapReduce model provides a simple, transparent way to parallelize the computation across the cluster.

3 Experiments

We have experimented with a number of experiments to evaluate the performance of the MapReduce model. We have run experiments on a variety of different machines and have observed that the MapReduce model is scalable and reliable in practice.

4 Related Work

There is a rich body of work on parallel computing and distributed systems. However, the MapReduce model provides a simple, efficient way to parallelize the computation across a large cluster of commodity servers. We have observed that the MapReduce model is scalable and reliable in practice.

5 Conclusion

We have observed that the MapReduce model is scalable and reliable in practice. We have run experiments on a variety of different machines and have observed that the MapReduce model is scalable and reliable in practice.

6 Acknowledgments

We would like to thank the many people who have contributed to the development of the MapReduce model. We would also like to thank the many people who have contributed to the development of the Google File System (GFS).
Bigtable: A Distributed Storage System for Structured Data

FAY CHANG, JEFFREY DEAN, SANJAY GHEMAWAT, WILSON C. HSIEH, DEBORAH A. WALLACH, MIKE BURROWS, TUSHAR CHANDRA, ANDREW FIKES, and ROBERT E. GRUBER
Google, Inc.

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this article, we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

Categories and Subject Descriptors: C.2.4 [Computer Communication Networks]: Distributed Systems—distributed databases

General Terms: Design

Additional Key Words and Phrases: Large-Scale Distributed Storage

ACM Reference Format:
MapReduce: A Flexible Data Processing Tool

MapReduce is a programming model for processing and generating large data sets. It allows users to specify a map function that processes a key-value pair and a reduce function that merges all intermediate values associated with the same intermediate key. We built a system around this programming model in 2003 to simplify construction of the inverted index for handling searches at Google.com. Since then, more than 10,000 distinct programs have been implemented using MapReduce at Google, including algorithms for large-scale graph processing, text processing, machine learning, and statistical machine translation. The Hadoop open source implementation of MapReduce has been used extensively outside of Google by a number of organizations.

The key advantage over parallel databases is that MapReduce programs are written in a declarative style and express the computation they wish to perform, not the specific machine where it should be run. This allows them to be moved to new machines and used in different ways without much effort. In this paper, we describe some of the key insights that led to the design of MapReduce and provide an overview of the system.
MapReduce is a programming model and an associated system for processing large data sets on a cluster of commodity hardware. Google's MapReduce was designed specifically to take advantage of the cluster environment by effectively breaking the problem into simple pieces and distributing them across the cluster. It was designed to make the job seem like a single-process job by hiding the complexity of the hardware configuration from the programmer. This paper describes the implementation of MapReduce at Google and the experiences that we have had with it.
HADOOP 1.0

Pig (data flow)
Hive (sql)
Others (cascading)

MapReduce
(cluster resource management & data processing)

HDFS
(redundant, reliable storage)
Fig. 3.3. (a) Bit cell of an all-parallel CAM, with iterative circuit implementation of word comparison (mismatch, $M_{ij}$) and addressed readout ($R_{ij}$).

(b) Bit cell of an all-parallel CAM, with wired-AND implementation of word comparison (match, $M_j$) and addressed readout ($R_j$).

- $A_i$ = address signal
- $C_j(1)$ = compare with 1
- $M$ = match output
- $W_i(1)$ = write 1
- $C_j(0)$ = compare with 0
- $H$ = mismatch out
- $W_i(0)$ = write 0
- $B_j$ = output data
Fig. 3.3. (a) Bit representation of word
(b) Bit cell of an arithmetic comparison (match)

Fig. 3.7a,b. Generation of the address of the uppermost response:
a) one module, b) modular tree (three levels)

Fig. 3.8. Shift-register implementation of the multiple-response resolver
The Connection Machine

Most computers have a single processing unit. In this new parallel computer 65,536 processors work on a problem at once. The resulting speed may transform several fields, including artificial intelligence

by W. Daniel Hillis

In the past three decades remarkable changes have taken place in digital computers. The amount of computational power that once took days to perform can now be done in seconds. Yet in certain fundamental respects the design of the digital computer remained unchanged between the days of the transistor (one of the first electronic switches) and the minicomputer built at the University of Pennsylvania in the late 1940s and the current generation of supercomputers. Most modern computers—from supercomputers to microprocessors—are similar to the transistor in that the memory and the central processing unit are separate entities. For a computation to be performed, the appropriate data must be retrieved from memory and brought to the central processor; the result is stored on the memory before being returned to memory.

Such a design is called sequential because the processing operations are performed one at a time. The sequential design was adopted mainly for utilitarian reasons. In the early days of digital computing the memory and the central processing unit were made of different materials. Since memory was slower than processing, it was desirable to maximize the efficiency of the processing unit at the expense of the memory’s efficiency. And that is just what the sequential design does. Today, however, the memory and the central processor are fabricated from the same silicon wafers. In a typical computer more than 90 percent of the silicon is devoted to memory. While the central processor is less than a hundreth the size of a wafer, memory, mainly large and idle. At about 11 million transistors per inch, packed into a silicon chip, the cost of memory is a significant expense to the processor.

Clearly, the general solution to this problem is to find a way to unify processing capacity and memory. But how? One answer is to exploit many small processors, working simultaneously, each accompanied by a small memory of its own. In such a design, which is called parallel processing, memory capacity and processing capacity can both be utilized with high efficiency. This is the approach my colleagues and I have taken in building a parallel computer called the Connection Machine. The Connection Machine incorporates 65,536 simple processors. Each processor is much less powerful than a typical personal computer, but in tandem they can execute several billion instructions per second, a rate that makes the Connection Machine one of the fastest computers ever constructed.

Yet the most interesting thing about the Connection Machine is not its brute speed but its flexibility. Special-purpose devices have been built that exploit parallelism to perform specific tasks quickly. Like idle servants, however, such machines are usually quite awkward outside their specialties. In contrast, the Connection Machine can operate at its peak processing rate in a wide range of applications. As this article will describe, the key to such flexibility is a communication network that enables the multitude of processors to exchange information in a pattern best suited to the problem at hand. The Connection Machine is not just a prototype. A dozen working Connection Machines are already commercialized, and they have begun to change the way digital computing treats problems in physics, image processing, text retrieval, and even artificial intelligence.

In order to understand the benefits of parallel processing, it is helpful to think about the difference between the way a conventional computer deals with an image and the way the image is treated in the human brain. From the pair of two-dimensional images falling on the retina a human brain is able—without apparent effort—to reconstruct a three-dimensional model of the world and maintain that model as the two-dimensional images change rapidly. Computers can be programmed to carry out part of the task, but even quite fast computers take hours to do what the human brain can do in fractions of a second [see “Visions by Man and Machine,” by Tomaso Poggio, Scientific American, April, 1984]. The brain maintains its advantage in spite of the fact that its components—neurons—are apparently millions of times slower than the computer’s transistors.

Why, then, is the brain so much faster than the computer? The visual circuitry of the brain is not fully understood, but it is clear that in some areas of the brain the principles of parallel processing are at work. In those parts of the brain the entire image is presented at once. The computer, however, examines the image one pixel at a time, as if it were looking through a minute keyhole. In the computer the image is represented as an array of numbers, each of which corresponds to the intensity of the light at a particular point. A typical low-resolution array might be a square with 256 pixels on a side. A conventional computer would need only one operation to compute the sum of all these pixels. The Connection Machine, on the other hand, assigns a single processor to each point of the image. Since every processor performs the same calculations on sets of 16 points simultaneously, a calculation that takes one second on an entire image is as fast as a calculation involving only a single point. For example, to find all the high points in the image that are brighter than a certain minimum, a sequential machine must check the 65,536 pixels each time the machine is ready to operate on a new set of 16 points.
The Innovator's Dilemma
When New Technologies Cause Great Firms to Fail

CLAYTON M. CHRISTENSEN
Disruptive Technologies: Catching the Wave

by Joseph L. Bower and Clayton M. Christensen

As the most consistent patterns in business are the failure of leading companies to stay at the top of their industries when technologies or market change. Geely and Ford missed the evolutionary, small-car market quite late. Xerox let Canon create the small-copier market. Bucyrus-Erie allowed Caterpillar and Dozer to take over the mechanical excavator market. Sears gave way to Wal-Mart.

The pattern of failure has been especially striking in the computer industry: IBM dominated the mainframe market but missed the emergence of mini-computers, which were technologically much simpler than mainframes. Digital Equipment dominated the minicomputer market with innovations like its VAX architecture but missed the personal-computer market almost completely. Apple Computer led the world of personal computing and established the standard for user-friendly computing but lagged five years behind the leaders in bringing its portable computer to market.

Why is it that companies like these invest aggressively and successfully in the technologies necessary to retain their current customers but then fail to make certain other technological investments that customers of the future will demand? Undoubtedly, bureaucracy, arrogance, taxed executive blood, poor planning, and short-term investment horizons have all played a role. But a more fundamental reason lies at the heart of the paradox: leading companies succumb to one of the most popular, and valuable, management dogmas: They stay close to their customers.

Although most managers like to think they are in control, customers wield extraordinary power in directing a company’s investments. Before managers decide to launch a technology, develop a product, build a plant, or establish new channels of distribution, they must look to their customers first. Do their customers want it? How big will the market be? Will the investment be profitable? The more astute managers ask and answer these questions...
Fig. 5. Cellular robot controller. Sensors on the robot (RB) detect lamp input (LI) and transmit a signal to the computer (PC). The signals from the sensors are recorded by the PC into a spatial light pattern which is projected as light stimuli with a video projector (PR) via mirror (MR) onto the surface of the plasmodium (PP). Oscillations of the plasmodium (PP), which is patterned on an agar plate (AP) by a plastic sheet (PS), are detected by a CCD camera (CC) as intensity changes in light transmitted from the bandpass filtered (BF) light source (LS).

Fig. 1. Plasmodium of Physarum polycephalum. The early stage (A) and in developed form, approximately 8 h later (B). In both panels the length of the white bar corresponds to 10 mm.
Robot control with biological cells

Soichiro Tsuda, Klaus-Peter Zauner, Yukio-Pegio Gunji

Graduate School of Science and Technology, Department of Earth & Planetary Science, Kobe University, Nada, Kobe 657-8501, Japan
School of Electronics and Computer Science, University of Southampton, SO17 1BJ, United Kingdom

Received 28 February 2005; received in revised form 8 July 2006; accepted 15 July 2006

Abstract

At present there exists a large gap in size, performance, adaptability and robustness between natural and artificial information processors for performing coherent perception-action tasks under real-time constraints. Even the simplest organisms have an enviable capability of coping with an unknown dynamic environment. Robots, in contrast, are still clumsy if confronted with such complexity. This paper presents a bio-hybrid architecture developed for exploring an alternate approach to the control of autonomous robots. Circuits prepared from amoeboid plasmodia of the slime mold Physarum polycephalum are interfaced with an omnidirectional hexapod robot. Sensory signals from the macro-physical environment of the robot are transduced to cellular scale and processed using the unique micro-physical features of intracellular information processing. Conversely, the response form the cellular computation is amplified to yield a macroscopic output action in the environment mediated through the robot’s actuators. © 2006 Elsevier Ireland Ltd. All rights reserved.

Keywords: Autonomous robots; Molecular computing; Coupled oscillators; Biologically inspired robotics

1. Introduction

The prevalent approach to robot control is based on ‘behaviour decomposition’. The interaction loop between robot and environment is decomposed and treated as individual modules. This approach simplifies controller design but places significant burden on each individual module. The use of ‘behaviour decomposition’ enables robots to successfully work in either a stable work space or with the support of a teacher. The scheme can be implemented in localised (Tani and Nolfi, 1999) or distributed fashion (Tani et al., 2004). Without a teacher, however, delineating a boundary for the environment...
Thank You
Thank UoL