Karakterisering en synthese van werklasten voor datacenteroptimalisatie

Workload Characterization and Synthesis for Data Center Optimization

Stijn Polfliet

Promotor: prof. dr. ir. L. Eeckhout
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Computerwetenschappen

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. J. Van Campenhout
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2012 - 2013

UNIVERSITEIT
GENT

ISBN 978-90-8578-588-0
NUR 980
Wettelijk depot: D/2013/10.500/21

Acknowledgements

Several people have contributed in one way or another to the creation of this dissertation. I wish to thank them for their supportive help. Specific thanks go out to:

- My advisor prof. Lieven Eeckhout for giving me the opportunity to be part of the Performance Lab team, for supporting me in the best possible way and still leaving room to find my own way.
- The members of the examination commission of this dissertation: prof. Luc Taerwe, prof. Koen De Bosschere, prof. Filip De Turck, dr. Stijn Eyerman and dr. Rudy Van Hoe for their valuable suggestions to improve this dissertation. Special thanks to prof. Benjamin C. Lee and prof. Yiannakis Sazeides for taking the time to visit our University and being part of the commission.
- Frederick Ryckbosch, colleague and friend, for the past decade of joining forces.
- Nicolas Eenaeme, and the other people from MassiveMedia, who were so enthusiastic to help us bridging the gap between research and industry.
- My colleagues from the Computer Systems Lab for the wonderful, funny and inspiring moments, during work, lunch or other events.
- My parents, my brother and sister, my family, for giving me the opportunity to realize my dreams.
- And, of course, Kaat, for always being there for me.

Examination commission

Prof. Luc Taerwe, **chairman**
Prodean Faculty of Engineering and Architecture
Ghent University

Prof. Koen De Bosschere, **secretary**
ELIS Department
Faculty of Engineering and Architecture
Ghent University

Prof. Lieven Eeckhout, **advisor**
ELIS Department
Faculty of Engineering and Architecture
Ghent University

Prof. Filip De Turck
INTEC Department
Faculty of Engineering and Architecture
Ghent University

Prof. Benjamin C. Lee
Electrical and Computer Engineering
Duke University
USA

Prof. Yiannakis Sazeides
Department of Computer Science
University of Cyprus
Cyprus

Dr. Stijn Eyerman
ELIS Department
Faculty of Engineering and Architecture
Ghent University

Dr. Rudy Van Hoe
Server and Cloud Platform
Microsoft Benelux

Samenvatting

Het wereldwijde web is geëvolueerd van één statische webpagina in 1990 tot een uitdeinend universum van 40 miljard webpagina's. In de begindagen van het internet werden er vooral statische, informatieve webpagina's aangeboden. Vandaag bieden de meeste webapplicaties complexe diensten aan zoals e-mail, e-handel, mediatoepassingen, kantoortoepassingen en sociale netwerkdiensten. Dit heeft voor een grondige verschuiving van computergebruik aan de klantzijde naar computergebruik aan de serverzijde gezorgd — men noemt dit vaak 'cloud computing'.

Deze uiteenlopende internetdiensten hebben samen met de opkomst van smartphones en tablets tot de snelle evolutie van het wereldwijde web geleid. Het aantal personen dat gebruikmaakt van tal van internetdiensten is de laatste decennia zeer snel gestegen. Om dit groot aantal gebruikers te ondersteunen is er een degelijke infrastructuur nodig. De dimensionering van deze infrastructuur, servers en datacenters, zorgt voor een aantal uitdagingen, omwille van de volgende redenen.

1. Hedendaagse webapplicaties hebben duizenden tot miljoenen gebruikers. Om een groot aantal gebruikers te kunnen ondersteunen worden grootschalige datacenters gebouwd. Deze datacenters zijn zeer kostgevoelig: indien we de kostprijs van een server kunnen verlagen met slechts enkele tientallen euro's, kan dit een grote kostbesparing betekenen op het niveau van het datacenter. Er zijn verschillende factoren die de totale kostprijs van een datacenter beïnvloeden, bv. hardware-infrastructuur, voeding- en koelinginfrastructuur, software, operationele kosten, herstellingen, personeel en gebouwen.

2. Een ander belangrijk aspect dat de rendabiliteit van een webapplicatie bepaalt, is klantentevredenheid. Dit wordt meestal uitgedrukt in responstijd en service-niveauovereenkomst[1]. Deze maatstaven worden gebruikt om uit te drukken hoe lang een gebruiker moet wachten alvorens de webpagina geladen is. Eerder onderzoek heeft aangetoond dat er een rechtstreeks verband is tussen de laadtijd van een webpagina en verkoopsomzet.

[1] Eng.: Service-Level Agreement (SLA)

Het doel van deze thesis is om de prestatie en totale kostprijs van een datacenter waarin hedendaagse webapplicaties draaien zoveel mogelijk te reduceren, terwijl er voldaan wordt aan service-niveauovereenkomsten en terwijl de responstijd binnen aanvaardbare grenzen blijft. Om dit te bewerkstelligen is er een gedetailleerde karakterisering nodig van de werklasten die draaien binnen het datacenter. Bovendien is het belangrijk om de impact te schatten van deze werklasten op de totale kostprijs van het datacenter.

Het dimensioneren van de energiedistributie en de koelinginfrastructuur in een datacenter gebeurt standaard op basis van het vermogenverbruik dat wordt aangegeven door de technische fiche van de server. Deze waarden worden door de hardwarefabrikanten met een ruime marge meegegeven, wat echter leidt tot een overgeprovisioneerd datacenter. Een meer kostenefficiënte manier is om een datacenter te dimensioneren op basis van het maximale vermogenverbruik dat effectief ooit verbruikt zal worden. In dit onderzoek stellen we een methodologie voor om op een automatische manier zogenaamde vermogenvirussen te genereren, dit zijn werklasten die het vermogenverbruik voor een bepaald platform maximaliseren.

Door rekening te houden met het soort werklast dat effectief in het datacenter zal gebruikt worden, is het mogelijk om extra winst te boeken op het vlak van prestatie en kostenbesparingen. We presenteren in dit onderzoek een studie van een aantal datacentrische werklasten en hun impact op de totale kostprijs van een datacenter.

We gaan vervolgens nog een stap verder door een datacenter te optimaliseren voor één specifieke werklast. Omdat sociale-netwerksites tegenwoordig zeer populair zijn, werd er gekozen om een grote sociale-netwerksite te analyseren en te bekijken hoe de totale kostprijs en de prestatie worden beïnvloed door verschillende keuzes in de dimensionering van de serverhardware. Omdat sociale-netwerksites zeer interactief zijn, wordt er extra aandacht besteed aan de prestatie, in functie van de responstijd zoals die ervaren wordt door de eindgebruiker.

Tijdens het optimaliseren van bovenvermelde webapplicatie is het duidelijk geworden dat het dupliceren van de volledige werklast een moeilijke taak is. In het bijzonder is het belangrijk om privacy en bedrijfsgevoelige informatie te beschermen. Daarom stellen we tenslotte in deze thesis een raamwerk voor om op een automatische manier een synthetische kloon van een databank op te stellen.

We onderscheiden de volgende bijdragen in deze thesis.

Multi-kern vermogenvirussen. Het energie- en vermogenverbruik zijn belangrijke factoren bij de dimensionering van grootschalige datacenters. Het energieverbruik is van groot belang voor het uitbouwen van het energiedistributienetwerk en voor het voorzien van de nodige koelinginfra-

structuur. Traditioneel wordt data uit de technische fiche van een server gebruikt om te bepalen hoeveel vermogen maximaal verbruikt zal worden door het betreffende platform. Deze gegevens zijn echter overschat en komen niet overeen met wat het serverplatform effectief maximaal zal verbruiken.

In deze thesis stellen we een raamwerk voor om op een automatische manier vermogenvirussen te genereren. Deze vermogenvirussen zijn programma's die het vermogenverbruik van een multi-kern serverplatform proberen maximaliseren. We gebruiken dit raamwerk om de impact te bestuderen van het dimensioneren van een datacenter op basis van een maximaal vermogenvirus in plaats van op basis van wat de fabrikant ons meegeeft. Onze resultaten tonen aan dat het vermogenverbruik drastisch kan verminderd worden.

Datacentrische werklasten. De hoeveelheid data die op het internet wordt geproduceerd, groeit zeer snel. Daarom richten we ons vervolgens op data-explosie en datadiversiteit. Er zijn een groot aantal verschillende datacentrische werklasten die in de achtergrond van een datacenter kunnen draaien, zoals bv. indexering, compressie, encryptie, audio- en videomanipulatie, enz. Het is belangrijk om deze datacentrische werklasten te karakteriseren en het datacenter te optimaliseren zodat we een zo hoog mogelijke prestatie per dollar kunnen behalen.

In dit werk bestuderen we hoe de opkomende klasse van datacentrische werklasten de dimensionering van een datacenter beïnvloedt. Via architecturale simulatie in een gevalideerde simulator leiden we enkele interessante inzichten af. De belangrijkste conclusie is dat het meest optimale platform afhankelijk is van het type werklast. Deze observatie suggereert dat heterogeniteit een oplossing kan bieden bij het verhogen van de efficiëntie. In een heterogeen datacenter wordt een bepaalde werklast uitgevoerd op het hardwareplatform dat het meest geschikt is. Onze resultaten tonen een verbetering van respectievelijk 88%, 24% and 17% in kostefficiëntie ten opzichte van een homogeen datacenter met high-end, standaard of low-end hardware.

Karakterisering van sociale-netwerksite. Omwille van het grote aantal gebruikers en de grootschaligheid van de infrastructuur is het dimensioneren van een datacenter voor hedendaagse sociale-netwerksites een grote uitdaging.

In deze thesis presenteren we een casestudie waarbij we een grootschalige sociale-netwerksite karakteriseren en waarbij we verschillende hardware- en softwarekeuzes maken. De werklast wordt zowel in de tijd als in de ruimte bemonsterd om een werklast te bekomen waarvan het haalbaar is om een aantal experimenten mee te doen. De gereduceerde werklast bevat

alle belangrijke diensten (en hun interacties) en laat toe om verschillende hardwarekeuzes af te wegen en hun impact op de prestatie te meten.

We beschouwen de Netlog werklast, een populaire commerciële socialenetwerksite met een groot aantal gebruikers. We verkennen verschillende hardware-opties met betrekking tot het aantal processorkernen, processorklokfrequentie, traditionele harde schijven versus flashgebaseerde schijven, enz. Onze resultaten tonen bv. dat het gebruik van een flash-gebaseerde harde schijf in de databankserver de langste responstijden met 30% verlaagt.

Synthetische databanken. Het databanksysteem is een belangrijk onderdeel van een hedendaagse Web 2.0 werklast. De prestatie ervan verbeteren is een grote uitdaging. Het is niet eenvoudig om hardware- en softwarealternatieven te vergelijken in een productieomgeving en bovendien is het niet altijd mogelijk om een databank volledig of deels te kopiëren omwille van privacyredenen of omwille van bedrijfsgevoelige informatie.

Daarom stellen we een raamwerk voor om op een automatische manier een synthetische kloon van een databank te maken. Dit wordt gerealiseerd door eerst een statistisch profiel op te stellen van de originele databank. Vervolgens wordt er een synthetische databank geconstrueerd op basis van dit statistisch profiel. De synthetische variant vertoont dezelfde statistische eigenschappen als de originele databank, maar verbergt gebruikers- en bedrijfsgevoelige informatie. Deze techniek laat toe om de prestatie te analyseren in een aparte omgeving, eventueel zelfs door andere partijen.

We evalueren het voorgestelde raamwerk gebruikmakend van de databank van de sociale-netwerksite Netlog. We tonen aan dat het statistisch profiel en de synthetische databankkloon in beperkte tijd gegenereerd kunnen worden. Bovendien valideren we de nauwkeurigheid ten opzichte van de originele databank en concluderen we dat er een fout is van 0.95% en 1.38% respectievelijk voor het gemiddelde en 90-percentiel van de responstijden. De techniek kan gebruikt worden om zowel hardware- als softwarekeuzes te bestuderen, bv. het effect van de processorklokfrequentie, het gebruik van traditionele harde schijven versus flashgebaseerde schijven, enz.

Summary

The World Wide Web has evolved from a single static Web page in 1990 into a growing universe of 40 billion Web pages as of today. Early Internet services provided mostly static, informational Web pages. Today, most Web applications offer complex services like e-mail, e-commerce, rich media applications, office applications and social networking, leading to a dramatic shift from client-side computing to server-side computing, often referred to as cloud computing.

These various novel Internet services that are being offered, along with ubiquitous Internet access possibilities through various devices including mobile devices such as smartphones and tablets, have led to this fast growth of the World Wide Web. In particular, smartphones enable their users to be permanently in touch with e-mail, the Internet, social networking sites such as Facebook and Twitter, e-commerce, etc.

Hence, the number of people using Internet services of various kinds is increasing rapidly and demonstrates the large scale of applications and systems behind these services. Designing the servers and data centers to support contemporary Internet services is challenging, for a number of reasons.

1. Current Web applications have thousands and up to millions of users, which requires distributed applications running in large data centers. The ensemble of servers is often referred to as a warehouse-scale computer and scaling out to this large a scale clearly is a major design challenge. Because of their scale, data centers are very much cost driven — optimizing the cost per server even by only a couple tens of dollars results in substantial cost savings and proportional increases in profit. There are various factors affecting the cost of a data center, such as the hardware infrastructure (servers, racks and switches), power and cooling infrastructure, software, operational expenses, repairs, management personnel, and real estate. Hence, data centers are very cost-sensitive and need to be optimized for the ensemble. As a result, operators drive their data center design decisions towards a sweet spot that optimizes performance per dollar.

2. Another important aspect that determines the profitability of a Web

application, is customer satisfaction. This is usually expressed in end-user response time and service-level agreements, i.e., how long a user has to wait before the Web page is loaded. Previous research has shown that every increase in page load time is directly related to sales revenue.

The central focus of this dissertation is to optimize the total cost of ownership (TCO) of a data center running contemporary Internet applications. Our goal is to reduce TCO of the data center as much as possible while delivering service-level agreements and keeping response times within acceptable bounds, thus securing company profit. This involves a detailed characterization of the workloads in the data center, and their impact on TCO.

Common practice is to (over-)provision the power and cooling infrastructure of a data center by using nameplate power consumption, as provided by hardware vendors, which leads to severe cost-inefficiency. A more cost-efficient approach is to consider achievable maximum power consumption numbers and dimension the data center towards this number rather than nameplate power. We therefore present a methodology for automatically generating so-called power viruses that maximize power consumption.

There is an additional opportunity to optimize both cost and performance by dimensioning the data center for the Internet services that are actually running in the data center. We present a case study with an emerging class of data-centric workloads and evaluate how these workloads affect design decisions in the data center.

We then go yet one step further and optimize the data center for a single workload. Since social networking applications are prevalent these days, we focus on a large networking application and analyze how data center design decisions affect cost and application performance. Because social networking applications are highly interactive, we focus on how data center design decisions affect user-perceived performance in terms of response times.

One particular challenge we faced when optimizing the data center for one Web application is the ability to duplicate the entire workload to a testing environment, including databases, web server software, etc. One important concern with this approach is to guarantee security and privacy while anonymizing the data in the duplicated database. We address these issues in our final research work on synthetic databases by presenting a framework for automatically generating a synthetic clone of an existing database.

More specifically, this dissertation makes the following contributions.

Multi-core power viruses. Energy and power usage are key design concerns in servers and large-scale data centers, for several reasons. Energy-related costs, for both empowering and cooling the servers, have become an important component in the total cost of ownership of this class of systems. To improve the energy and cost efficiency of servers and data centers, we need appropriate tools to understand power usage at the system level.

This work proposes a framework for automatically generating full-system multi-core powermarks, or power viruses, which are synthetic programs with desired power characteristics on multi-core server platforms. We use this framework to study the impact on the total energy cost of dimensioning the power supply units at the power usage determined by a max powermark rather than the nameplate power consumption which potentially leads to a significant reduction in power consumption.

Data-centric workloads. The amount of data produced on the Internet is growing rapidly. We therefore next focus on data explosion and the trend towards more and more diverse data, including rich media such as audio and video. Data explosion and diversity leads to the emergence of data-centric workloads to manipulate, manage and analyze the vast amounts of data. These data-centric workloads are likely to run in the background and include application domains such as data mining, indexing, compression, encryption, audio/video manipulation, data warehousing, etc.

Given that data centers are very much cost-sensitive, reducing the cost of a single component by a small fraction immediately translates into huge cost savings because of the large scale. Hence, when designing a data center, it is important to understand data-centric workloads and optimize the ensemble for these workloads so that the best possible performance per dollar is achieved.

This work studies how the emerging class of data-centric workloads affects design decisions in the data center. Through the architectural simulation of minutes of run time on a validated full-system x86 simulator, we derive the insight that for some data-centric workloads, a high-end server optimizes performance per total cost of ownership (TCO), whereas for other workloads, a low-end server is the winner. This observation suggests heterogeneity in the data center, in which a job is run on the most cost-efficient server. Our experimental results report that a heterogeneous data center achieves an up to 88%, 24% and 17% improvement in cost-efficiency over a homogeneous data center with high-end, commodity and low-end servers, respectively.

Real-life Web 2.0 workload characterization. Designing data centers for Web 2.0 social networking applications is a major challenge because of the large number of users, the large scale of the data centers, the distributed ap-

plication base, and the cost sensitivity of a data center facility. Optimizing the data center for performance per dollar is far from trivial.

In this dissertation, we present a case study characterizing and evaluating hardware/software design choices for a real-life Web 2.0 workload. We sample the Web 2.0 workload both in space and in time to obtain a reduced workload that can be replayed, driven by input data captured from a real data center. The reduced workload captures the important services (and their interactions) and allows for evaluating how hardware choices affect end-user experience (as measured by response times).

We consider the Netlog workload, a popular and commercially deployed social networking site with a large user base, and we explore hardware trade-offs in terms of core count, clock frequency, traditional hard disks versus solid-state disks (SSD), etc., for the different servers, and we obtain several interesting insights.[2] For example, our experiments show that using an SSD reduces the longest response times by 30% over a regular HDD in the database servers.

Synthetic database cloning. The database management system is an important component of a contemporary Web 2.0 workload, yet improving its performance is challenging. Evaluating hardware and software alternatives and trade-offs in a production environment is complicated and might not always be possible; copying (part of) a database to an offline environment might not be feasible either, particularly because of intellectual property and privacy issues.

In the fourth contribution, we propose a framework for generating synthetic but representative database clone workloads. This is done by computing a statistical profile of the original database, and by subsequently generating a synthetic database from this statistical profile. The synthetic database exhibits the same statistical properties as the original database but obfuscates and anonymizes business and user information. The key benefit of this approach is that it enables performance analyses of a representative, yet anonymized database workload in an offline environment. Synthetically generated database workloads even allow for running performance analyses at third-party sites, and answering what-if questions regarding database scalability with respect to user count and/or novel service features.

We evaluate the proposed framework using the database from the real-life Web 2.0 Netlog workload. We demonstrate that a statistical profile and synthetic database workload can be generated in limited time, and we validate the synthetic clone to be representative for the original workload, i.e., we report a 0.95% and 1.38% error for predicting the mean and the 90% percentile of the response time distribution, respectively, using the syn-

[2] http://www.netlog.com

thetic workload compared to the original workload. The synthetic clone is anonymized by construction. We illustrate the usefulness of synthetically generated database clones for driving both hardware and software trade-offs, including exploring the effect on response time of CPU clock frequency, hardware prefetching, hard drive versus solid-state disk, alternative database storage engines, and database size scaling.

Contents

List of Figures

List of Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
AMD	Advanced Micro Devices
API	Application Programming Interface
BBV	Basic Block Vector
BIC	Bayesian Information Criterion
CMP	Chip-level Multiprocessing
CPU	Central Processing Unit
DBMS	Database Management System
DRAM	Dynamic random-access memory
DSS	Decision Support Systems
ERP	Enterprise Resource Planning
FAWN	Fast Array of Wimpy Nodes
GNU	GNU's not Unix!
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
HW	Hardware
ILP	Instruction-Level Parallelism
I/O	Input/output
ISA	Instruction-set architecture
IT	Information Technology
JVM	Java Virtual Machine
L1	Level-1 Cache
L2	Level-2 Cache
L3	Level-3 Cache
LLC	Last Level Cache
MIPS	Millions of Instructions Per Second
NIC	Network Interface Card
OLTP	Online Transaction Processing
OOO	Out-of-order
OSDB	Open Source Database Benchmark
PARSEC	Princeton Application Repository for Shared-Memory Computers
PCAP	Packet Capture
PCI	Peripheral Component Interconnect

PHP	Hypertext Preprocessor
RDBMS	Relational Database Management System
ROB	Reorder Buffer
SIMD	Single Instruction Multiple Data
SMT	Simultaneous multithreading
SPEC	Standard Performance Evaluation Corporation
SQL	Structured Query Language
SSD	Solid State Disk
SSE	Streaming SIMD Extensions
TCO	Total Cost of Ownership
TDP	Thermal Design Power
TPC	Transaction Processing Performance Council
URL	Uniform Resource Locator

Chapter 1

Introduction

The World Wide Web is moving to its 25th anniversary. Over time, the Web has evolved from a single static Web page in 1990, over 26 Web pages by the end of 1992 into a growing universe of 40 billion Web pages as of today.[1] Internet usage has grown by a factor of 6.6x over the past twelve years worldwide according to a recent study by Internet World Stats.[2] The Internet-sector server market is growing at a steady pace as well, i.e., IDC studies show that the high-performance computing server market is expected to grow 8% yearly until 2015.[3] As another example, worldwide revenue from public IT cloud services is expected to grow by 27.6% by 2015.[4]

Early Internet services provided mostly static, informational Web pages. Today, most Web applications offer complex services like e-mail, e-commerce, rich media applications, office applications and social networking, leading to a dramatic shift from client-side computing to server-side computing, often referred to as cloud computing.

These various novel Internet services that are being offered, along with ubiquitous Internet access possibilities through various devices including mobile devices such as smartphones and tablets, have led to this fast growth of the World Wide Web. In particular, smartphones enable their users to be permanently in touch with e-mail, the Internet, social networking sites such as Facebook and Twitter, e-commerce, etc. There are over one billion smartphones worldwide today, and the next billion is expected to be achieved by 2015.[5]

Hence, the number of people using Internet services of various kinds is increasing rapidly and demonstrates the large scale of applications and

[1] http://www.worldwidewebsize.com/

[2] http://internetworldstats.com/stats.htm

[3] http://www.idc.com/getdoc.jsp?containerId=prUS23386912

[4] http://www.idc.com/prodserv/idc_cloud.jsp

[5] http://blogs.strategyanalytics.com/WDS/post/2012/10/17/Worldwide-Smartphone-Population-Tops-1-Billion-in-Q3-2012.aspx

systems behind these services. For example, there are more than 1 billion people using Facebook actively each month; 600 million Facebook users use mobile devices and these users are twice as active as non-mobile users — according to Facebook's statistics as of November 2012.[6] As another example, there are more than 500 million registered Twitter users generating more than 100 million Twitter messages a day, as of July 2012.[7]

1.1 Key challenges

Designing the servers and data centers to support contemporary Internet services is challenging, for a number of reasons. As mentioned above, current Web applications have thousands and up to millions of users, which requires distributed applications running in large data centers [5]. The ensemble of servers is often referred to as a warehouse-scale computer [6] and scaling out to this large a scale clearly is a major design challenge. Because of their scale, data centers are very much cost driven — optimizing the cost per server even by only a couple tens of dollars results in substantial cost savings and proportional increases in profit. There are various factors affecting the cost of a data center, such as the hardware infrastructure (servers, racks and switches), power and cooling infrastructure, software, operational expenses, repairs, management personnel, and real estate. Hence, data centers are very cost-sensitive and need to be optimized for the ensemble. As a result, operators drive their data center design decisions towards a sweet spot that optimizes performance per dollar.

Another important aspect that determines the profitability of a Web application, is customer satisfaction. This is usually expressed in end-user response time and service-level agreements i.e., how long a user has to wait before the Web page is loaded. Previous research at Amazon has shown that every increase in page load time of 100 ms decreases sales by 1% [42]. Similarly, Google has reported that moving from a 10-result page loading in 0.4 seconds to a 30-result page loading in 0.9 seconds decreased traffic and ad revenues by 20%.[8]

1.2 Thesis Contributions

The central focus of this dissertation is to optimize the total cost of ownership (TCO) of a data center running contemporary Internet applications. Our goal is to reduce TCO of the data center as much as possible while

[6] http://newsroom.fb.com/

[7] http://semiocast.com/publications/2012_07_30_Twitter_reaches_half_a_billion_accounts_140m_in_the_US

[8] http://glinden.blogspot.be/2006/11/marissa-mayer-at-web-20.html

delivering service-level agreements and keeping response times within acceptable bounds, thus securing company profit. This involves a detailed characterization of the workloads running in the data center, and their impact on TCO.

Common practice is to (over-)provision the power and cooling infrastructure of a data center by using nameplate power consumption, as provided by hardware vendors, which leads to severe cost-inefficiency. A more cost-efficient approach is to consider achievable maximum power consumption numbers and dimension the data center towards this number rather than nameplate power. We therefore present a methodology for automatically generating so-called power viruses that maximize power consumption.

There is an additional opportunity to optimize both cost and performance by dimensioning the data center for the Internet services that are actually running in the data center. We present a case study with an emerging class of data-centric workloads and evaluate how these workloads affect design decisions in the data center.

We then go yet one step further and optimize the data center for a single workload. Since social networking applications are prevalent these days, we focus on a large networking application and analyze how data center design decisions affect cost and application performance. Because social networking applications are highly interactive, we focus on how data center design decisions affect user-perceived performance in terms of response times.

One particular challenge we faced when optimizing the data center for one Web application is the ability to duplicate the entire workload to a testing environment, including databases, web server software, etc. One important concern with this approach is to guarantee security and privacy while anonymizing the data in the duplicated database. We address these issues in our final research work on synthetic databases by presenting a framework for automatically generating a synthetic clone of an existing database.

More specifically, this dissertation makes the following contributions.

Contribution 1: Multi-core power viruses

Energy and power usage are key design concerns in servers and large-scale data centers, for several reasons. Energy-related costs, for both empowering and cooling the servers, have become an important component in the total cost of ownership of this class of systems. To improve the energy and cost efficiency of servers and data centers, we need appropriate tools to understand power usage at the system level.

This work proposes a framework for automatically generating full-system multi-core powermarks, or power viruses, which are synthetic programs with desired power characteristics on multi-core server platforms. We use this framework to study the impact on the total energy cost of dimensioning the power supply units at the power usage determined by a max powermark rather than the nameplate power consumption which potentially leads to a 63% reduction in power consumption. The framework is additionally used to construct full-system power models with error bounds on the power estimates, and to guide the design of energy-efficient and cost-efficient server and data center infrastructures.

This work on power viruses has been published in:

> Stijn Polfliet, Frederick Ryckbosch and Lieven Eeckhout, "Automated Full-System Power Characterization", In *IEEE Micro*, Vol. 31, No. 3, 46-59, May/June 2011.

Contribution 2: Data-centric workloads

The amount of data produced on the Internet is growing rapidly. We therefore focus next on data explosion and the trend towards more and more diverse data, including rich media such as audio and video. Data explosion and diversity leads to the emergence of data-centric workloads to manipulate, manage and analyze the vast amounts of data. These data-centric workloads are likely to run in the background and include application domains such as data mining, indexing, compression, encryption, audio/video manipulation, data warehousing, etc.

Given that data centers are very much cost-sensitive, reducing the cost of a single component by a small fraction immediately translates into huge cost savings because of the large scale. Hence, when designing a data center, it is important to understand data-centric workloads and optimize the ensemble for these workloads so that the best possible performance per dollar is achieved.

This work studies how the emerging class of data-centric workloads affects design decisions in the data center. Through the architectural simulation of minutes of run time on a validated full-system x86 simulator, we derive the insight that for some data-centric workloads, a high-end server optimizes performance per total cost of ownership (TCO), whereas for other workloads, a low-end server is the winner. This observation suggests heterogeneity in the data center, in which a job is run on the most cost-efficient server. Our experimental results report that a heterogeneous data center achieves an up to 88%, 24% and 17% improvement in cost-efficiency over a homogeneous data center with high-end, commodity and low-end servers, respectively.

This work on data-centric workloads has been published in:

> Stijn Polfliet, Frederick Ryckbosch and Lieven Eeckhout, "Optimizing the Data Center for Data-Centric Workloads", In *Proceedings of the International Conference on Supercomputing (ICS)*, 182-191, June 2011.

Contribution 3: Real-life Web 2.0 workload characterization

Designing data centers for Web 2.0 social networking applications is a major challenge because of the large number of users, the large scale of the data centers, the distributed application base, and the cost sensitivity of a data center facility. Optimizing the data center for performance per dollar is far from trivial.

In this work, we present a case study characterizing and evaluating hardware/software design choices for a real-life Web 2.0 workload. We sample the Web 2.0 workload both in space and in time to obtain a reduced workload that can be replayed, driven by input data captured from a real data center. The reduced workload captures the important services (and their interactions) and allows for evaluating how hardware choices affect end-user experience (as measured by response times).

We consider the Netlog[9] workload, a popular and commercially deployed social networking site with a large user base, and we explore hardware trade-offs in terms of core count, clock frequency, traditional hard disks versus solid-state disks (SSD), etc., for the different servers, and we obtain several interesting insights. For example, our experiments show that using an SSD reduces the longest response times by 30% over a regular HDD in the database servers. Further, we present two use cases illustrating how our characterization method can be used for guiding hardware purchasing decisions as well as software optimizations.

This work on a real-life Web 2.0 workload has been published in [61]:

> Stijn Polfliet, Frederick Ryckbosch and Lieven Eeckhout, "Studying Hardware and Software Trade-Offs for a Real-Life Web 2.0 Workload", In *Proceedings of the Third Joint WOSP/SIPEW International Conference on Performance Engineering (ICPE)*, 181-192, 2012.

Contribution 4: Synthetic database cloning

The database management system is an important component of a contemporary Web 2.0 workload, yet improving its performance is challenging. Evaluating hardware and software alternatives and trade-offs in a production environment is complicated and might not always be possible; copying

[9] http://www.netlog.com

(part of) a database to an offline environment might not be feasible either, particularly because of intellectual property and privacy issues.

In this work, we propose a framework for generating synthetic but representative database clone workloads. This is done by computing a statistical profile of the original database, and by subsequently generating a synthetic database from this statistical profile. The synthetic database exhibits the same statistical properties as the original database but obfuscates and anonymizes business and user information. The key benefit of this approach is that it enables performance analysis of a representative, yet anonymized database workload in an offline environment. Synthetically generated database workloads even allow for running performance analyses at third-party sites, and answering what-if questions regarding database scalability with respect to user count and/or novel service features.

We evaluate the proposed framework using the database from the real-life Web 2.0 Netlog workload. We demonstrate that a statistical profile and synthetic database workload can be generated in limited time, and we validate the synthetic clone to be representative for the original workload, i.e., we report a 0.95% and 1.38% error for predicting the mean and the 90% percentile of the response time distribution, respectively, using the synthetic workload compared to the original workload. The synthetic clone is anonymized by construction. We illustrate the usefulness of synthetically generated database clones for driving both hardware and software trade-offs, including exploring the effect on response time of CPU clock frequency, hardware prefetching, hard drive versus solid-state disk, alternative database storage engines, and database size scaling.

This work on synthetic databases is described in:

> Stijn Polfliet, Frederick Ryckbosch and Lieven Eeckhout, "Performance Analysis through Database Workload Cloning", In *ACM Transactions on Architecture and Code Optimization (TACO)*, Accepted (under revision) for publication.

1.3 Overview

This dissertation is organized as follows.

We present a framework for automatically generating full-system multi-core powermarks in Chapter 2. We use this framework to construct full-system power models with error bounds on the power estimates, and to guide the design of energy-efficient and cost-efficient server and data center infrastructures.

In Chapter 3, we characterize data-centric workloads running as back-

ground tasks in the data center. Through the architectural simulation of minutes of run time on a validated full-system x86 simulator, we derive the insight that the optimal hardware platform depends on the specific data-centric workload setting.

In Chapter 4, we present a case study on a real-life Web 2.0 workload. We sample the Web 2.0 workload both in space and in time to obtain a reduced workload that can be replayed, driven by input data captured from a real data center. We present some insights in how the data center should be optimized for this particular workload.

In Chapter 5, we present a framework for generating synthetic but representative database clone workloads. The key benefit of this approach is that it enables performance analyses of a representative, yet anonymized database workload in an offline environment.

Chapter 6 concludes this dissertation with a summary and discussion of future research directions.

Chapter 2

Powermark Benchmarks

Energy and power usage are key design concerns in servers and large-scale data centers. Current practice in data center design is to dimension the power grid and cooling based on the server's nameplate power. However, nameplate power is typically overestimated by hardware vendors, leading to a highly overprovisioned data center. Determining maximum achievable power consumption is challenging. We therefore present a framework for automatically characterizing maximum achievable power consumption in the data center by constructing full-system multi-core powermarks, or synthetic programs which maximize power consumption on multi-core server platforms.

2.1 Introduction

Energy and power usage are key design concerns in servers and large-scale data centers, for a number of reasons. Energy-related costs, for both empowering and cooling servers, have become an important component in the total cost of ownership (TCO) of this class of systems. In fact, with electricity costs trending up, energy-related costs will become an increasingly larger part of the total cost [6]. In addition, as we become more and more environmentally conscious, improving the energy efficiency of computer systems is key for reducing carbon dioxide emissions by the IT industry. Finally, increased power density and temperature may lead to reduced reliability of the hardware. In order to improve the energy-efficiency and cost-efficiency of servers and data centers, we need appropriate tools to understand power usage at the system level.

We present a framework to automatically characterize power consumption at the system level in server hardware. The proposed framework allows for automatically generating so-called powermarks, or synthetic programs with specific power characteristics. For example, particular powermarks may strive to maximize power consumption at a given CPU uti-

lization level through a realizable program. In contrast to prior work in this area which built CPU-centric single-threaded stressmarks [23] [38], the proposed framework constructs full-system multi-core powermarks: next to stressing the CPU, the powermarks also stress main memory, network I/O as well as disk I/O. Further, the framework is versatile to the hardware platform of interest as it generates powermarks in a high-level programming language. An evaluation on six hardware platforms illustrates the ability to generate powermarks that exceed the power consumed by a large set of performance benchmarks and existing so-called torture tests.

In a subsequent step, we explore two applications for the powermarks framework. First, we use it to automatically construct full-system power models that provide an estimate for a server's total power consumption at a given utilization level. The model's key asset is that it provides meaningful error bounds on the predicted power consumption. The error bounds are determined by powermarks that maximize and minimize power usage through a realizable program. We demonstrate the model's accuracy and applicability on six hardware platforms, and we show that the models can handle both Simultaneous Multithreading (SMT) and Chip Multiprocessors (CMP) architectures.

Second, we use powermarks to guide data center design and dimensioning. In one case study, we study the impact on the total energy cost of dimensioning the power supply units at the power usage determined by a max powermark rather than the 'nameplate' power consumption. Depending on the hardware platform, we report savings in energy cost ranging between 34% and 63%. Going one step further, we evaluate the idea of power capping at a pre-set power usage level below the max powermark. This is done by monitoring the power consumption at the server's power outlet and feeding this back to the power manager running on the server. If power consumption is about to exceed the pre-set power cap, the power manager artificially reduces the server's utilization level by putting the server to sleep at regular intervals so that the effective power usage stays below the power cap. We demonstrate a case study illustrating the effectiveness of power capping with limited impact on overall performance.

This chapter is organized as follows. We first elaborate on the powermark generation framework, followed by a description of two potential applications, automated power modeling and server/data center dimensioning. Finally, we discuss related work and conclude.

2.2 Powermark framework

There exists substantial prior work in stressmark generation. Several sources [21] [25] [76] report how industry builds stressmarks, often called

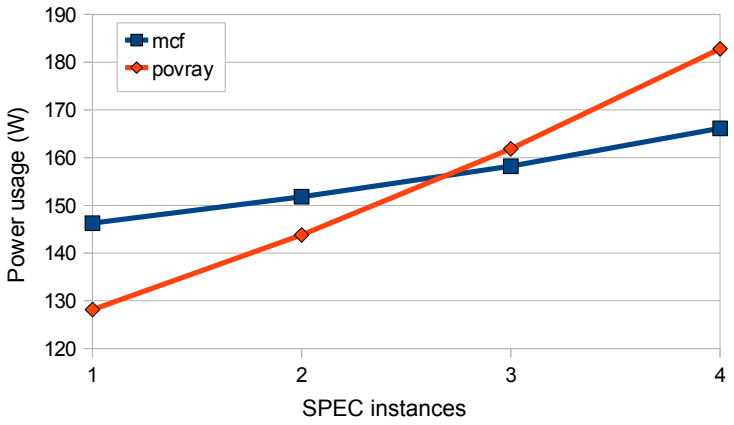

Figure 2.1: Average power consumption as a function of the number of co-run instances for the SPEC CPU2006 mcf and povray benchmarks on the AMD quad-core processor system.

power viruses, that maximize CPU power consumption, temperature, dI/dt, etc. Building stressmarks is typically done manually, and is therefore very time-consuming, tedious and error-prone. To speed up the time-consuming process of stressmark generation, recent work proposed frameworks for automatically generating stressmarks while focusing on the CPU [38], both the CPU and memory [23] and multi-core CPUs [9] [41]. None of these approaches provide full-system stressmarks though, as they focus on the CPU and memory only. Our powermark framework aims at generating powermarks that stress the entire system, including the network interface and the disk alongside the CPU and memory. In addition, our framework targets multi-core systems.

In order to illustrate that generating multi-core stressmarks is a non-trivial extension to single-core stressmarks, we first make a number of observations before describing our framework in more detail. A first observation is that a benchmark that yields higher power consumption than another benchmark when run in isolation does not necessarily yield higher power consumption when multiple instances co-run; see Figure 2.1 for the example benchmarks mcf and povray on our baseline AMD quad-core Opteron system. We observed similar results for other benchmarks. (See later for a detailed description of the experimental setup.) This suggests that a powermark that is optimized for a single core does not necessarily imply max power consumption for a multi-core processor, and hence, a powermark generator needs to co-optimize per-core power consumption and overall chip power consumption. The reason is that contention in shared multi-core resources affects per-core performance and power con-

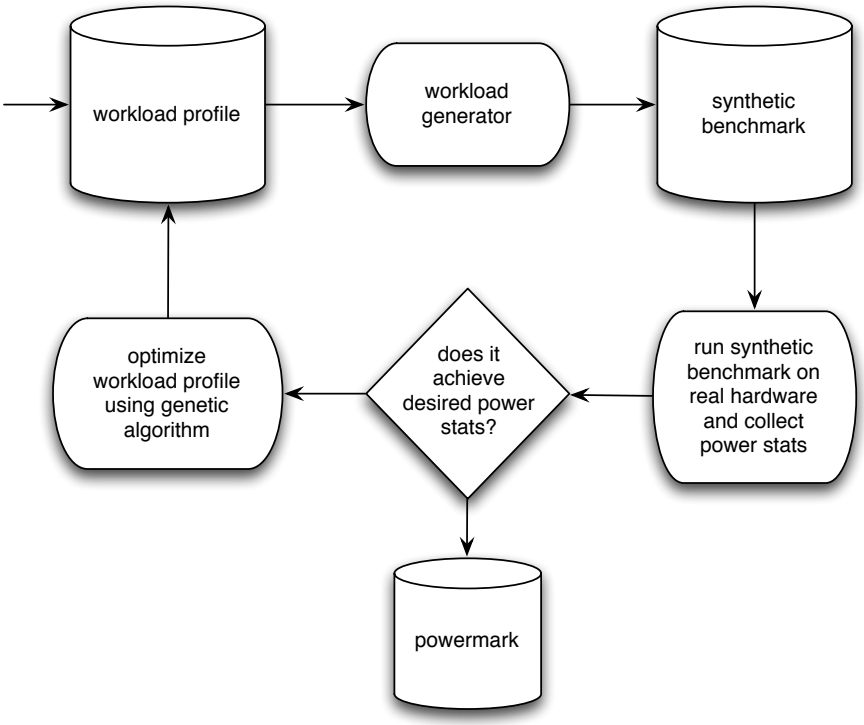

Figure 2.2: The powermark framework.

sumption.

A second observation worth noting is that running multiple copies of the same workload does not yield maximum power consumption. Memory accesses to shared state by co-executing threads yield substantially higher power consumption due to cache coherency traffic. In our framework, we found that memory accesses to shared state increase power consumption substantially, up to 4.5 Watt (or 1.9% relative to max power) when comparing powermarks that include versus exclude shared state memory accesses on our baseline system. These two observations imply that a multi-core powermark is not achieved by simply co-executing multiple instances of a single-core powermark, and, in addition, there are complex interactions among co-executing threads through shared memory and through shared resources. Further, powermarks are platform-specific, hence, an automated approach to constructing powermarks is desirable. This motivates for a framework that can explore this space effectively and efficiently.

2.2.1 Framework overview

Figure 2.2 illustrates our powermark framework. The central piece in the framework is a *workload generator* that generates a synthetic benchmark from a set of workload characteristics. The synthetic benchmark is built from a program skeleton; the workload characteristics give the synthetic benchmark its behavioral characteristics of interest. Because finding the workload characteristics that maximize or minimize power consumption is non-trivial because of complex interactions between workload behavior and the target hardware, we employ a genetic algorithm to automatically evolve towards a powermark with desired power characteristics. The genetic algorithm starts off with a number of randomly generated workload profiles, each of which collects a number of workload characteristics. Each workload profile serves as input to a workload generator, which in its turn generates a synthetic benchmark from it with the desired workload characteristics. Each synthetic benchmark is then run on real hardware (or a simulator) and power statistics are collected. If the measured power statistics match the desired power statistics for at least one of the synthetic benchmarks, we have found our powermark and the optimization process ends — the synthetic benchmark is our powermark. If not, we yield to the genetic algorithm to explore the workload space and try out new workload profiles.

We now discuss the three steps in the framework: workload generator, collection of power stats, and automated exploration.

2.2.2 Powermark generator

The workload generator takes a workload profile as input and generates a synthetic benchmark. The synthetic benchmark follows a particular skeleton with a number of parameterized workload characteristics. The goal of the genetic algorithm then is to search the set of parameterized characteristics that make the resulting powermark satisfy the desired power characteristics.

The synthetic benchmark skeleton consists of a number of compute-intensive threads along with one disk thread and one network thread. The network thread sends out network packets with a parameterized sleep time between packets. The disk thread similarly reads at random locations on the disk with parameterized sleep time between disk reads. The compute-intensive threads can be configured through a number of parameters, such as the number of threads, the size of memory accessed by an individual thread only, the size of memory accessed by all threads to invoke cache coherency traffic, the strides with which the program traverses these regions of memory, the instruction mix, and the inter-instruction dependencies to

Characteristic	Description
network thread sleep time	time between network packets
disk sleep time	time between disk reads
no. of threads	number of CPU-intensive threads
loop size	number of instructions in the main loop of the CPU-intensive thread
size of thread-local memory	size of memory region accessed by an individual thread
size of thread-shared memory	size of memory region accessed by all threads
memory stride profile	stride and probability for traversing memory regions
dependency distance	minimum dependency distance counted as the number of dynamically executed instructions between two dependent instructions
program instruction mix	percentage arithmetic, memory and branch operations
memory mix profile	percentage load-local, load-shared, store-local, store-shared
arithmetic mix profile	percentage instruction type out of 18 types including integer, floating-point (single-precision, double-precision), SSE instructions
branch transition rate	transition rate and probability

Table 2.1: Workload characteristics that serve as input to the synthetic benchmark generator.

control Instruction-Level Parallelism (ILP). The reason for having separate disk and network threads is to be able to stress both the disk and the network while stressing the CPU and memory at the same time. We run the powermark for 30 seconds at least.

A CPU thread consists of a loop of instructions over which it iterates, see also Figure 2.3. The instructions in the loop are determined by the characteristic dimensions, as we explain below. For each instruction in the loop, we determine its type, its dependencies, its memory stride access pattern (in case of a load or store) and its branch behavior (in case of a branch). There are 43 characteristics in total, subdivided in 12 categories, see also Table 2.1. The workload generator uses so-called characteristic dimensions for the CPU threads. Each of these dimensions consists of a number of value-probability pairs, and the sum of these pairs equals one within a dimension. We explain each of these dimensions in more detail now.

- The program mix dimension consists of three value-probability pairs: one set for the probability of arithmetic operations, one for memory operations, and one for branch operations. This dimension lays out the instruction mix of a CPU thread.
- The arithmetic mix dimension holds 18 value-probability pairs, each representing one type of arithmetic instruction, i.e., integer addition, floating-point multiplication, SIMD, division, etc.
- The dependency distance dimension represents what the likelihood is for observing a particular inter-instruction dependency distance in the synthetic benchmark; dependency distance is defined as the number of dynamically executed instructions between a write and a read

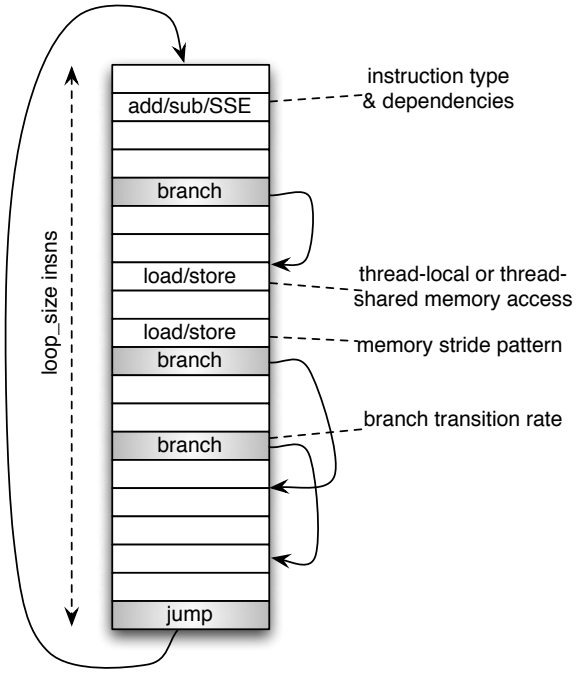

Figure 2.3: The loop body of the CPU-intensive thread(s).

to a register or memory location. We keep track of 20 dependency distances.

- The memory mix dimension holds 4 value-probability pairs, with each pair representing one of the following: load operations on thread-local memory, store operations on thread-local memory, load operations on thread-shared memory, or store operations on thread-shared memory.
- The memory stride dimension determines the stride value with which (local or shared) memory is traversed, i.e., a stride value of s means that if memory is read at location A, the next read will be at location $A + s$. Multiple stride values can be specified with respective likelihoods of occurrence.
- The branch dimension represents the branch transition rate [32], or the number of times a branch switches between taken and not-taken, divided by the number of times the branch is executed. Multiple transition rates can be specified with corresponding likelihoods of occurrence.

As mentioned before, we generate synthetic benchmarks in a high-level programming language (C in our case), and not assembly. The reason for

doing so is to be able to more easily deploy our framework across platforms. Generating synthetic benchmarks with specific behavioral characteristics comes with a number of challenges that one would not encounter when generating synthetic benchmarks at the assembly level. One issue relates to generating SIMD instructions such as SSE and its further enhancements in the x86 architecture. In order to force the compiler to generate SIMD instructions — which we found to be a significant contributor to the overall CPU power consumption (1.9 Watt per core on our baseline) — we make use of vectorizable data types. Further, we generate C code such that the compiler cannot optimize the code away through dead code elimination or copy propagation. This is done by letting each thread return a value to the main thread that it computes based on all the computed values in the thread. Finally, we want the desired code and branch characteristics as we generate the synthetic benchmark to be preserved after compilation. Therefore, we force the compiler to allocate variables in registers (by using the `-O1` flag and not `-O0`), and not to do loop hoisting or if-conversion — this is done by using specific compiler flags when compiling the powermarks. For the same reason, we randomize all variables at the start of the execution of the synthetic benchmark in order to avoid constant folding and copy propagation by the compiler.

2.2.3 Power monitoring

The next step is to run the synthetic benchmark on real hardware or on a simulator. In our setup, we run the synthetic benchmark on real hardware platforms and we measure power consumption at the power outlet. This is done using the Racktivity RC0816 device [62], which measures power usage in real-time. Since the power measurement device is connected to a server's power outlet, it measures full-system power consumption. The Racktivity power monitoring device measures power consumption at a one second time granularity within 0.1 Watt of accuracy. We measure maximum (or minimum) power consumption observed during the course of the entire program execution. Room temperature is kept constant at 24 degrees Celsius. We measured measurement variability across multiple runs, and we found the variability to be less than 0.1%.

2.2.4 Automated exploration

To drive the search process we employ a genetic algorithm, which is well known to be effective in avoiding local optima. The genetic algorithm searches the workload space by varying the workload characteristics in the abstract workload description, and optimizes these characteristics towards a powermark. As mentioned before, the genetic algorithm starts from a

random set of so-called workload profiles. A workload profile characterizes a workload through its characteristic dimensions. The collection of these workload profiles is called a generation; there are 20 workload profiles in our setup. These workload profiles are evaluated according to the objective function, also called the fitness function, i.e., a synthetic benchmark is generated and is run on real hardware and power stats are collected. A new population, an offspring, which is a subset of these workload profiles, is probabilistically selected. The likelihood for a workload profile to be selected is determined by the workload profiles fitness functions, i.e., a fitter workload profile is more likely to be selected. The offspring consists of 10 workload profiles. We also retain the single-best workload profile across generations. Selection alone cannot introduce new workload profiles in the search space, therefore mutation and crossover are performed to build the next generation of 20 workload profiles. Crossover is performed, with probability p_{cross}, by randomly exchanging parts of two selected workload profiles from the current offspring. This means that we exchange a number of characteristic dimensions among two workload profiles to create new workload profiles. The mutation operator prevents premature convergence to local optima by randomly altering parts of a workload profile, with probability p_{mut}. Mutation changes some characteristic dimensions in a workload profile to create a new one. The generational process is continued until a specified termination condition has been reached. In our experiments we specify the termination condition as the point when there is little or no improvement in the objective function across successive generations or we have iterated 50 generations. In our setup, we set p_{cross} and p_{mut} to 0.4 and 0.2, respectively; these parameters were determined experimentally. The end result of the genetic algorithm is an abstract workload configuration for which the corresponding synthetic benchmark stresses the objective function the most — this is our powermark. In this work, we consider two objective functions, max power and min power.

2.3 Powermark evaluation

Evaluating powermarks is troublesome. The main difficulty is that one cannot prove that a powermark that is optimized to maximize power consumption effectively maximizes power consumption. In other words, one cannot prove that there does not exist some other program that consumes even more power than the powermark. This is true for both manually and automatically generated powermarks.

Nevertheless, to assess our framework, we compare the power consumed by the generated powermarks against a broad range of existing performance benchmarks (SPEC CPU2006 and PARSEC) as well as a set of

Torture test	Optimized for	Language
burnBX	stressing cache/memory interfaces	x86 assembly
burnK6	AMD K6	x86 assembly
burnK7	AMD K7	x86 assembly
burnMMX	stressing cache/memory interfaces	x86 assembly
burnP5	Intel Pentium	x86 assembly
burnP6	Intel Pentium Pro, II, II, Celeron	x86 assembly
MPrime	All CPUs, all ISAs	C

Table 2.2: Torture tests.

Processor	Full description	Nameplate power
Opteron Quad-Core	AMD Opteron Quad-Core 2350 2.0 GHz	380 W
Opteron Dual-Core	2x Dual-Core AMD Opteron 2212 HE	600 W
Athlon	AMD Athlon XP 3000+	350 W
Core i7	Intel Core i7 CPU 920 @ 2.67 GHz	500 W
Pentium 4	Intel Pentium 4 CPU @ 3.20 GHz	230 W
Atom	Intel Atom CPU 330 @ 1.60 GHz	60 W

Table 2.3: Hardware platforms.

so-called torture tests. When running the single-threaded SPEC CPU2006 benchmarks, we run as many copies of a benchmark as there are hardware thread contexts on the hardware platform. For the multi-threaded PARSEC benchmarks, we run as many threads as there are hardware contexts. The torture tests are single-threaded synthetic benchmarks designed to 'torture' or stress the hardware platform, see Table 2.2. We also run as many copies of a torture test as there are hardware thread contexts on the platform. We consider six hardware platforms in our evaluation, see Table 2.3. Figure 2.4 shows the power consumed by the various torture tests along with the maximum power consumed by one of the SPEC CPU2006 and PARSEC benchmarks (called 'others' in the legend). The torture tests generate the largest power consumption for four out of the six hardware platforms, albeit by a small margin only compared to the performance benchmarks. For the Intel Atom and Pentium 4, two SPEC CPU benchmarks (dealII and libquantum) exceed the torture tests.

Figure 2.5 quantifies by how much the powermarks exceed the maximum power consumed by the torture tests and the performance benchmarks shown earlier (labeled 'best power virus'). We consider two flavors of powermarks: (i) a CPU powermark that maximizes CPU and memory power but does not stress the other system components, and (ii) a full-system powermark that maximizes power consumption of the entire system (including the disk and the network). The CPU powermark increases maximum system power consumed by up to 4% for the Intel Core i7 and Atom platforms. The full-system powermark increases maximum power consumed over the CPU powermark by up to slightly less than 3% for the dual-socket AMD Opteron system. It is interesting to note that the largest

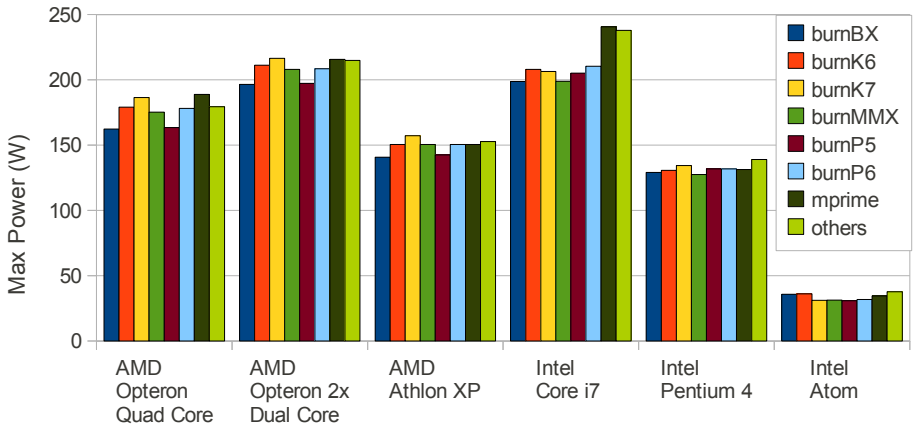

Figure 2.4: Power consumption for the torture tests versus SPEC CPU and PARSEC (labeled 'others') on six hardware platforms.

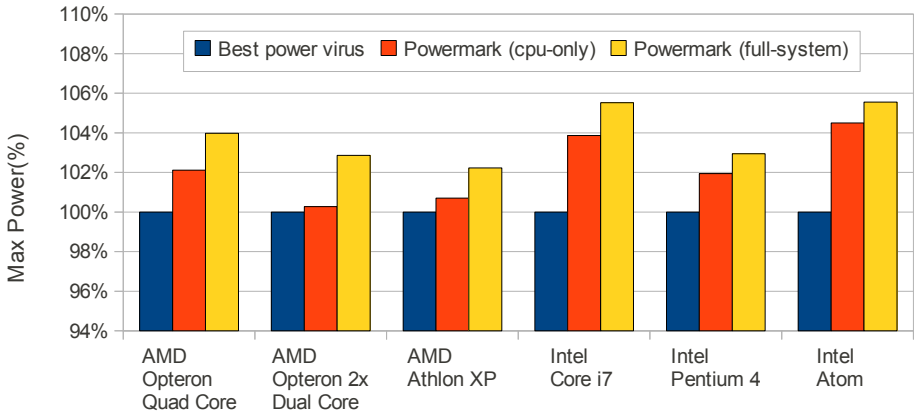

Figure 2.5: Power consumption of the powermarks versus the max torture test and performance benchmarks.

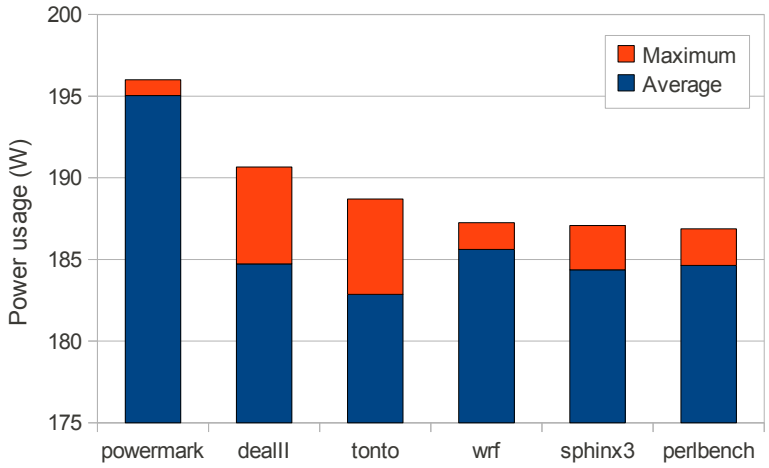

Figure 2.6: Average and maximum power consumption for the powermark compared to the top-5 power-hungry performance benchmarks on the AMD quad-core Opteron processor platform.

max power deltas are obtained for the most recent processors, namely the AMD Opteron, Intel Core i7 and Intel Atom. This is most likely due to aggressive clock gating in these processors: the workload has a significant impact on what logic is active at a given point in time, and hence maximizing processor activity maximizes power consumption. The older processors, the AMD Athlon XP and Intel Pentium 4, presumably do not implement such aggressive clock gating, and hence a program might not have as much impact on the amount of power the processor consumes. Although the increase in max power consumption may seem small in absolute terms between a CPU-centric powermark and a full-system powermark, we believe it is significant because the disk and network are powered on while running a CPU powermark, and reading from a disk increases power consumption by 3.1 Watts only in our setup.

It is insightful to take a look at how the average and maximum power consumption of the powermark compares to the top-5 power-hungry performance benchmarks, see Figure 2.6. Recall that our power monitor quantifies power consumption at a one-second time granularity. The maximum power consumption reported in this graph is the maximum power consumption observed at a one-second granularity over the course of a benchmark's entire execution; the average number is the average power consumption over the entire benchmark execution. The gap between the maximum and average power consumption is smaller for the powermark than for the performance benchmarks. This means that the powermark achieves a high power consumption over a long period of time whereas the performance benchmarks exhibits power peaks only (and the peaks are lower

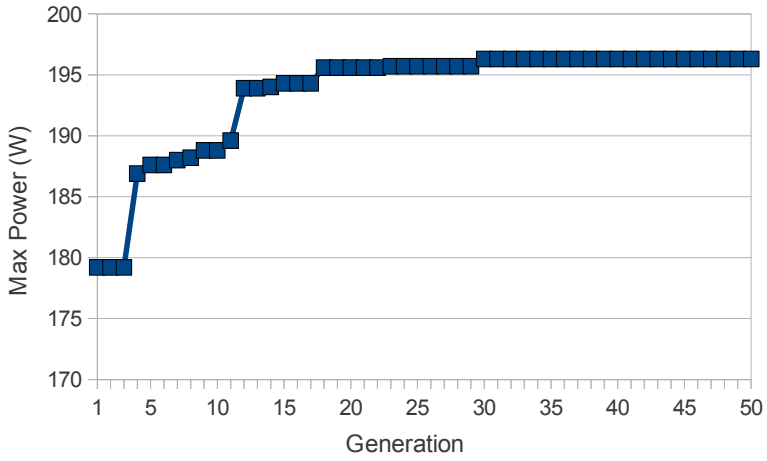

Figure 2.7: Convergence of the genetic algorithm on the AMD quad-core Opteron system.

than for the max powermark).

The powermark framework also provides an opportunity to study how workload characteristics affect power consumption. For illustrative purposes, we now compare the characteristics of the powermarks generated for the low-end Intel Atom versus the high-end Intel Core i7 processor system. We find that the Core i7 powermark exhibits 5 times more ILP compared to the Atom powermark, this means that the average dependency distance between instructions is 5 times larger. Furthermore, the Core i7 powermark accesses much larger thread-local and thread-shared spaces. This reflects the fact that the Core i7 is a wide superscalar out-of-order processor with large caches whereas the Atom is a narrow in-order processor with much smaller caches. Further, we observe a larger fraction of branches for the Core i7 powermark, whereas the Atom powermark exhibits a larger fraction of arithmetic and memory operations. Remarkably, the majority of the arithmetic operations are SIMD (SSE) operations: 70% on the Core i7 and 55% on the Atom. Further, the loads executed in the Core i7 powermark issue independent parallel memory accesses to stress the memory hierarchy to the fullest.

Finally, Figure 2.7 illustrates how fast the genetic algorithm converges. The randomly generated synthetic benchmarks at startup consume 179.2 Watt, whereas the powermark consumes 196.3 Watt, an increase by 17.1 Watt. The genetic algorithm converges to this maximum power consumption relatively fast in 30 generations. This evolutionary search took 10 hours of wall clock time.

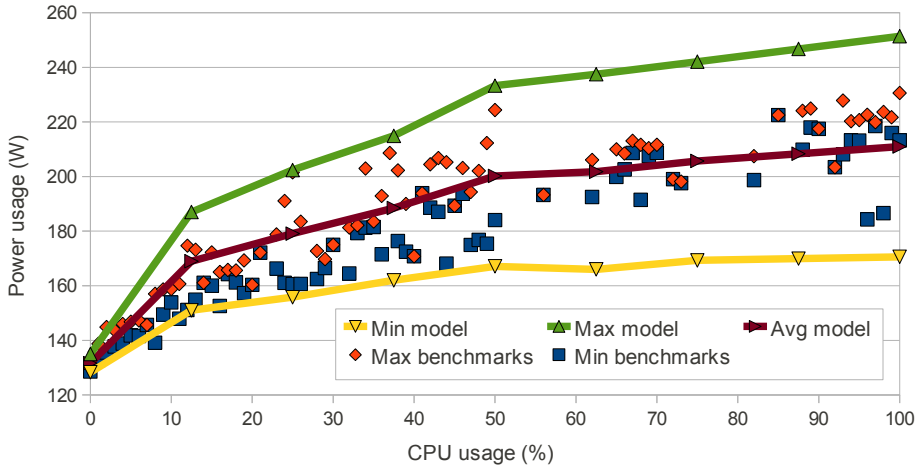

Figure 2.8: Validating the power model on the Intel Core i7 system.

2.4 Full-system power modeling

A first application that we explore for the powermark framework is to build system-level power models in an automated way. The idea is to generate two powermarks, one that maximizes power consumption and one that minimizes power consumption. We conjecture that the average between the maximum powermark and the minimum powermark corresponds to the average power consumption one would see across a large set of workloads. We found this to be true within 2.8%, see Figure 2.8. The vertical axis shows power consumption and the horizontal axis shows CPU usage. We consider both SPEC CPU2006 and PARSEC as our performance benchmarks. For SPEC CPU, we run 1-core, 2-core, 3-core and 4-core workloads by running multiple copies concurrently. For PARSEC, we run up to eight threads: we run each thread on an individual core for up to four cores, and beyond four threads, we schedule up to two threads per core through SMT execution. We report the maximum and minimum power consumption (at a one-second granularity) observed across those benchmarks. The performance benchmarks all fall within the bounds reported by the system-level power model. It is interesting to note that there is a knee in the curves at 50% CPU usage, or at 4 cores. This is because the Intel Core i7 has four cores with each core running two hardware SMT threads. Power consumption increases at a faster pace during multi-core execution relative to SMT execution.

Figure 2.9 shows similar results for two disk I/O intensive workloads, namely scp and tar: power consumption never exceeds the error bounds determined by the model. The key feature of this approach is that the model was constructed in an automated way and is easily applied on

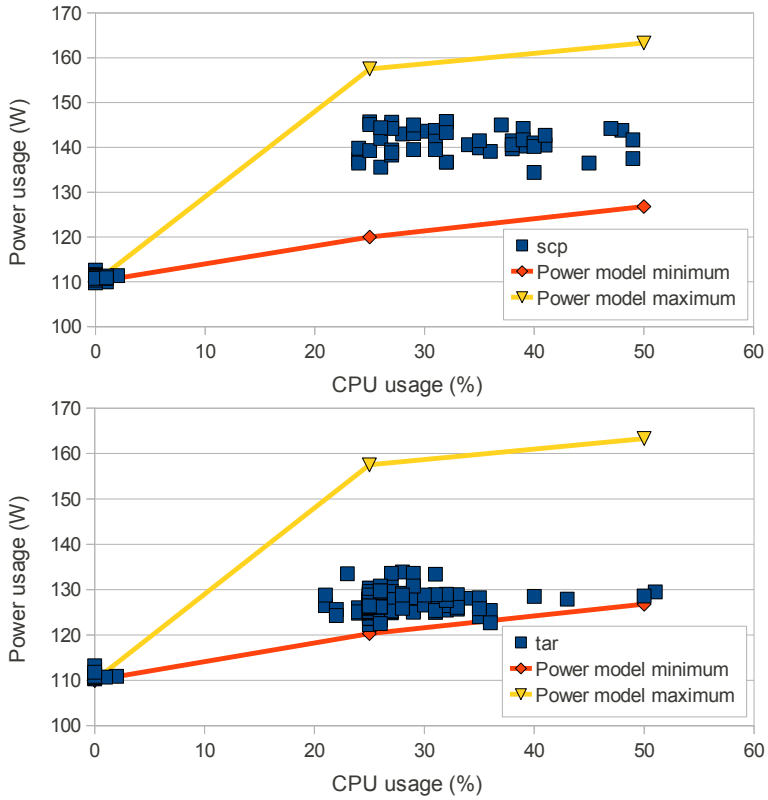

Figure 2.9: Validating the power model considering disk I/O intensive workloads: scp on the top and tar on the bottom on the AMD quad-core Opteron system.

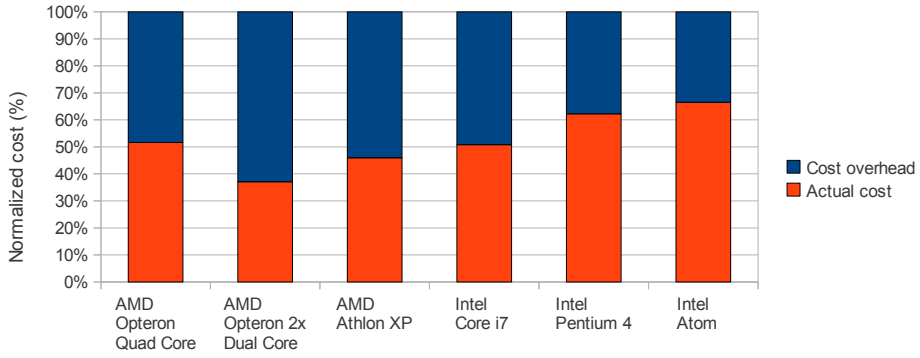

Figure 2.10: Normalized energy cost for a server dimensioned for the nameplate power consumption versus the max power consumption derived from a system-level powermark.

different platforms. In addition, the power model provides bounds that are determined in a systematic way using powermarks, in contrast to prior work which obtains error bounds through statistics on a set of workloads [20] [66].

2.5 Data center dimensioning

A second application that we explore is to dimension the data center using powermarks. The 'nameplate' power consumption of a server is typically overly high (worst case) compared to what the server is really consuming, e.g., the nameplate power consumption typically mentions what the server might consume with all the options installed, such as maximum memory, disk, PCI cards, etc. [6]. For example, the nameplate power consumption for the Intel Core i7 is rated 500 Watt, however, our system-level powermark consumes around 250 Watt. (See Table 2.3 for the nameplate power consumption for all of the hardware platforms.) This suggests that dimensioning the data center for the system-level powermark's power consumption rather than the nameplate power consumption can lead to significant cost reductions. In fact, some data center infrastructure providers charge their clients based on the nameplate power consumption and not the maximum realizable power consumption. The energy cost reduction by dimensioning on the maximum realizable power consumption is shown in Figure 2.10: normalized energy cost is shown as the normalized ratio of the nameplate power consumption to the powermark's power consumption. Dimensioning using the max powermark can reduce total energy cost between 34% and 63% depending on the hardware platform.

Going one step further, we realize that only a minority of the programs

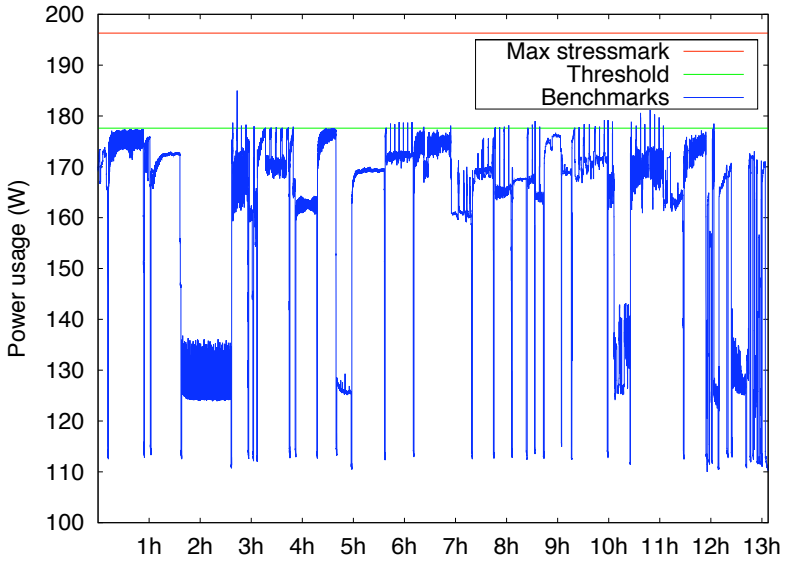

Figure 2.11: Dimensioning the server on a threshold power consumption 10% below the max powermark.

ever reach a power consumption level close to the max powermark. This suggests that it might be cost-efficient to dimension the system for a power consumption level below its max powermark, and have a monitoring and feedback system in place that lowers processor activity if actual power consumption exceeds the designed power level, so that in practice the system (almost) never exceeds the designed lower level. This reduces cost significantly while penalizing performance by a small margin only. We evaluated this idea using the user-space Linux `cpulimit` tool which allows for controlling the CPU usage of a process. In our experiment, we consider the AMD quad-core Opteron and we run all of the SPEC CPU2006 and PARSEC benchmarks one after another, and we set the threshold 10% below the max powermark, see Figure 2.11. The policy is such that if the actual power consumption exceeds the threshold — which it can for a short period of time given current power supplies — we run the processor at a usage level of 80% for 5 minutes. (Tuning the various parameters in this scheme is out of scope of this work.) The performance penalty is limited to 5.3%, while reducing cost by 10% compared to dimensioning with respect to the max powermark, or 58% in total compared to the nameplate power.

2.6 Related work

There are a number of research topics that the powermark framework relates to, which we discuss now.

Power viruses. Several sources report about power viruses for maximizing CPU power consumption [21] [25] [76]. These power viruses are developed manually which is time-consuming, tedious and error-prone. Recently, Joshi et al. [38] proposed a framework for automatically generating synthetic CPU power viruses from an abstract description of the workload; Ganesan et al. [23] generate power viruses that stress the CPU and memory. All of these approaches focus on the CPU and memory only, generate single-threaded power viruses, and do not stress the entire system. Further, transferring these power viruses from one platform to another is hard given that they are written in assembly. Our powermark environment on the other hand generates multi-threaded full-system powermarks and the framework is easily portable across platforms.

After publication of our original paper in IEEE Micro in 2011, several works continued along this line of research towards power viruses. More recently, Bertran et al. [9] and Kim et al. [41] focus on automatically building stressmarks for multi-core CPUs. This recent work only focuses on stressing the CPU, while our framework generates full-system powermarks that also stress other components like disks and network interfaces. Both papers present a framework that generates powermarks at assembly language level, while our framework generates powermarks in a high-level programming language. Generating powermarks in a high-level language makes it easier to use the framework on a wide range of hardware platforms.

Power benchmarking. Power benchmarking has received increased interest recently. In particular, SPEC released SPECpower_ssj2008 [45] which is a system-level, server-side Java workload that quantifies energy efficiency under varying loads. SPECpower generates and completes a mix of transactions and the reported throughput is the number of transactions completed per second over a fixed period of time; the workload considers 11 levels of load. Rivoire et al. [67] present JouleSort, a sort benchmark that reads its input from a file and writes its output to a file on a non-volatile device. There are three scale categories with 10 GB, 100 GB and 1 TB records, and the benchmark aims at covering multiple domains, from embedded, to mobile, as well as to the server domain. These power benchmarks do not strive at maximizing power consumption as we do with the powermark framework.

Power modeling. Power modeling has gotten significant interest over the past decade. Some efforts focus on simulation approaches for design exploration purposes, such as Wattch [13] and SimplePower [80]. Others have proposed power modeling approaches for real hardware. Isci et al. [35] for example use hardware performance counters for constructing a CPU-component power model. Fan et al. [20] use a more coarse-grain power model that builds on CPU utilization as the basis for modeling CPU power consumption. Rivoire et al. [66] proposed a framework for estimating full-system power consumption using CPU performance counters. The full-system power model developed in this work fits in the category of coarse-grain full-system models, and provides bounds on the power estimates. Meisner et al. [53] note that for the purpose of accurately predicting peak power consumption it is not sufficient to use average CPU utilization. The authors demonstrate that they can more accurately model peak power consumption by characterizing the relationship between server utilization and power supply behavior.

Synthetic benchmarks. Synthetic benchmarks have a long history, going back to Whetstone and Dhrystone which are manually crafted benchmarks that aimed at representing real workloads. Manually building benchmarks though is both tedious and time-consuming. More recent work focused on automatically generating synthetic benchmarks from an abstract workload description [8] [19] [37], the primary motivation being to speed up simulation by generating short-running synthetic benchmarks that are representative for long-running performance benchmarks. The powermark framework presented in this work builds on this body of work and generates synthetic benchmarks with specific power characteristics.

2.7 Conclusion

Power modeling is vital given today's focus on energy-efficiency. This work proposed a framework for automatically generating full-system powermarks, or synthetic benchmarks with specific power characteristics. The key novelty of the proposed framework is that it targets multi-core server hardware and that the powermarks stress the entire system; prior work focused on the CPU (and memory) only. The powermarks exceed power consumption of existing performance benchmarks and torture tests by a significant margin. Powermarks can be used for constructing full-system power models that are reasonably accurate, easy to develop and use, and provide error bounds; in addition, the powermarks can be used for dimensioning power provisioning in server and data center infrastructures.

Chapter 3

Data-centric Workloads

The amount of data produced on the Internet is growing rapidly, often referred to as 'big data'. Along with data explosion comes the trend towards more and more diverse data, including rich media such as audio and video. Data explosion and diversity leads to the emergence of data-centric workloads to manipulate, manage and analyze the vast amounts of data. These data-centric workloads are likely to run in the background and include application domains such as data mining, indexing, compression, encryption, audio/video manipulation, data warehousing, etc. Instead of only optimizing the data center for maximum power, we now focus on optimizing the data center for typical data-intensive applications, which enables further reductions in TCO.

3.1 Introduction

In this chapter we study how the emerging class of data-centric workloads affects design decisions in the data center. Through the architectural simulation of minutes of run time on a validated full-system x86 simulator, we derive the insight that for some data-centric workloads, a high-end server optimizes performance per total cost of ownership (TCO), whereas for other workloads, a low-end server is the winner. This observation suggests heterogeneity in the data center, in which a job is run on the most cost-efficient server. Our experimental results report that a heterogeneous data center achieves an up to 88%, 24% and 17% improvement in cost-efficiency over a homogeneous high-end, commodity and low-end server data center, respectively.

There are various factors affecting the cost of a data center, such as the hardware infrastructure (the servers as well as the rack and switch infrastructure), power and cooling infrastructure as well as operating expenditure, and real estate. Hence, warehouse-sized computers are very cost-sensitive, need to be optimized for the ensemble, and operators drive their

data center design decisions towards a sweet spot that optimizes performance per dollar. For example, commercial offerings by companies such as SeaMicro as well as ongoing research and advanced development projects such as the EuroCloud project, target low-end servers to optimize data center cost-efficiency.[1,2]

The emergence of warehouse-scale computers also leads to a dramatic shift in the workloads run on today's data centers. Whereas traditional data center workloads include commercial workloads such as database management systems (DBMS) and enterprise resource planning (ERP), the data centers in the cloud now run a new set of emerging workloads for online web services, e.g., e-commerce, webmail, video hosting, social networks. Users accessing these online Web services generate huge amounts of data, both text and rich media (i.e., images, audio and video). The workloads running on a warehouse-scale computer not only include the interactive interface with the end user but also distributed data processing and storage infrastructure. In addition, data analytics workloads need to run in the data center 'behind the scenes' to manage, manipulate, and extract trends from the vast amounts of online data. For example, an e-commerce application will feature a data mining workload running in the background to collect user profiles and make suggestions to its end users for future purchases. Similarly, web search engines feature indexing workloads running in the background to build up indices. Whereas traditional data center workloads are well studied historically, see for example [39] [49] [64], and online interactive workloads have emerged as a workload of interest in recent research efforts [1] [47] [65], data-centric workloads have received limited attention so far.

3.1.1 Data-centric workloads

We focus on the data-centric workloads that are likely to run as background processes in data centers in the cloud, i.e., workloads such as data mining, indexing, compression, encryption, rich media applications and data warehousing. We study how these data-centric workloads affect some of the design decisions in the data center. Through full-system simulations using a validated x86 simulator while simulating minutes of run time, we explore which server type optimizes the performance per dollar target metric for this set of emerging workloads. We conclude that there is no clear winner: for some workloads, a high-end server yields the best performance per cost ratio, whereas for others, a middle-of-the-road server is a winner, and for yet other workloads, a low-end server yields the best performance-cost efficiency.

[1] http://www.seamicro.com/

[2] http://www.eurocloudserver.com/

This result suggests the case for heterogeneous data centers in which a workload is run on its most performance-cost efficient server. For our set of workloads and experimental setup (which assumes equal weight for all workloads), a homogeneous low-end server data center improves performance-cost efficiency by 14% compared to a homogeneous high-end server data center; we report an 18% better performance-cost efficiency for a heterogeneous data center relative to a homogeneous data center with high-end servers only. We also observe that a heterogeneous data center with a collection of high-end servers and low-end servers achieves most of the benefits that can be achieved through heterogeneity; adding middle-of-the-road servers does not contribute much.

Obviously, the improvement achieved through heterogeneity very much depends on the workloads that co-execute in the data center. Considering a wide range of workload mixes, we report performance-cost efficiency improvements for a heterogeneous data center up to 88%, 24% and 17% compared to homogeneous high-end, commodity and low-end server data centers, respectively. Because estimating a data center's total cost of ownership is non-trivial, we also report results quantifying the performance-cost efficiency as a function of the cost ratio between the various server types, and by doing so, we determine the sweet spot for heterogeneous data centers. Finally, we present a comprehensive analysis to gain insight into where the benefit comes from. In the cases where the high-end server achieves a better performance-cost efficiency, the higher cost is offset by the higher throughput achieved through higher clock frequency, lower execution cycle counts and larger core counts. For the benchmarks for which the low-end processor is more performance-cost beneficial, the higher throughput achieved on the server is not offset by its higher cost.

We believe this is an interesting result given the current debate in the community on high-end versus commodity (middle-of-the-road) versus low-end servers for the data center [55] [59]. In particular, Lim et al. [47] conclude that lower-end consumer platforms and low-cost, low-power components from the high-volume embedded/mobile space may lead to a 2× improvement in performance per dollar. Reddi et al. [65] similarly conclude that a low-end Atom processor is more favorable than a high-end Intel Xeon for an industry-strength online web search engine, although these processors would benefit from better performance to achieve better quality-of-service and service-level agreements. In spite of these recent studies pointing towards low-end embedded servers for performance-cost efficient data centers, there is no consensus as to whether contemporary data centers should consider high-end versus low-end versus middle-of-the-road server nodes [55] [59]. Some argue for low-end 'wimpy' servers (see T. Mudge's statement in [55]) whereas others argue for high-end servers, and yet others argue for middle-of-the-road 'brawny'

servers (see U. Hölzle's statement in [55]). We conclude there is no single answer. For some workloads, high-end servers are most performance-cost efficient, whereas for other workloads, low-end embedded processors are most efficient.

3.1.2 Contributions and outline

This work makes the following contributions.

- We collect a set of data-centric workloads and we study how these workloads affect design decisions in the data center. Recent work in architectural studies for the data center considered online interactive workloads for the most part, and did not consider data-centric workloads. Running data-centric workloads requires minutes of run time on large data sets. We employ full-system simulation for doing so using a validated architectural simulator.

- We obtain the result that high-end and middle-of-the-road servers can be more cost-efficient than low-end servers for running data-centric workloads. This is in contrast to recent work, see for example [1] [47] [65], which argues for lower-end servers to optimize cost-efficiency and/or energy-efficiency in the data center. The reason for this outcome is that data-centric workloads are computation-intensive and frequency-sensitive, hence, high-end and middle-of-the-road servers yield a substantially better performance per cost ratio.

- We demonstrate that for some sets of data-centric workloads, a heterogeneous data center in which each workload runs on its most cost-efficient server, can yield significant cost savings.

- We provide detailed sensitivity analyses to gain insight in the benefits of heterogeneity and how it varies with workload mixes, server infrastructure cost and energy cost. In particular, we demonstrate that heterogeneity is beneficial for a range of cost ratios between a high-end versus a low-end server. Further, we demonstrate that the benefit from heterogeneity is higher at lower energy costs.

The remainder of this chapter is organized as follows. We first describe the data-centric workloads that we consider in this study (Section 3.2). We subsequently detail on the data center modeling aspects and our experimental setup (Section 3.3). We then describe our results (Section 3.4) and provide sensitivity analyses (Section 3.5). Finally, we discuss related work (Section 3.6) and conclude (Section 3.7).

3.2 Data-Centric Workloads

3.2.1 Data explosion and diversity in the cloud

A prominent trend that we observe in the cloud is data explosion, also referred to as 'big data'. The amount of online data has grown by a factor of 56× over 7 years, from 5 exabytes of online data in 2002 to 281 exabytes in 2009 — a substantially larger increase compared to Moore's law (16× over 7 years) [63]. The reason comes from the emergence of interactive Internet services (e.g., e-commerce, web mail) and Web 2.0 applications such as social networking (e.g., Facebook, Twitter), blogs, wikis, etc., as well as ubiquitous access to online data through various mobile devices such as netbooks and smartphones.

Along with data explosion comes the trend of increasingly diverse data, including structured data, unstructured data and semi-structured data. In addition, the data stored in Web 2.0 applications is increasingly rich media, including images, audio and video.

Data explosion and diversity preludes a novel area of data-centric workloads in the cloud to manipulate the data, manage this huge data volume, extract useful information from it, derive insight from it, and eventually act on it. Hence, it is important to study these workloads and understand how this emerging class of workloads may change how data centers are optimized for performance-cost efficiency.

3.2.2 A data-centric benchmark suite

Motivated by this observation, we collected a number of benchmarks to represent the emerging application domain of data-centric workloads. We identify a number of categories such as data mining, indexing, security, rich media, compression, and data warehousing. Each of these categories prelude important emerging applications in data-centric workloads. We select benchmarks for each of these categories, see also Table 3.1.

Data mining. Analyzing the data is absolutely crucial to gain insight from it and eventually act on it. This requires data mining, statistical analysis and machine learning to extract and understand the underlying phenomena. We include three data mining benchmarks, namely kmeans, eclat and hmmer. The kmeans benchmark is a clustering workload that discovers groups of similar objects in a database to characterize the underlying data distribution. Clustering algorithms are often used in customer segmentation, pattern recognition, spatial data analysis, etc. Our dataset includes 100 K data points in an 18-dimensional space and groups these points in 50 clusters. The eclat benchmark is a typical Association Rule Mining (ARM)

workload to find interesting relationships in large data sets (466 MB in our case). The benchmark tries to find all subsets of items that occur frequently in a database. The hmmer benchmark involves the pfam collection of multiple sequence alignments and hidden Markov models (HMM) covering many common protein domains and families. It is used for running the hmmpfam executable, part of the HMMER package. Its input is a sequence of 9,000 residues that is being compared against 2,000 HMMs.

Indexing. Analyzing the data often requires indexing the data to enable efficient searching. We include the Apache lucene text search engine. In our case, the lucene crawler builds an index for 50 K Wikipedia pages (647 MB in total). The lucene benchmark is a Java workload and runs on the Open JDK JVM v6.

Data compression. Storing huge volumes of data requires compression and decompression in order to be able to store the data on disk in an efficient way. Our benchmark suite includes the tarz application which consists of the standard GNU tar utility to create an archive from, in our case, a set of PDF and text files. The archive is compressed using gzip (GNU zip). Gzip reduces the size of the archive using Lempel-Ziv (LZ77) encoding. The uncompressed input equals 1.2 GB in size and is compressed to 273 MB.

Data security. Data stored in the cloud may be proprietary or personal, and third parties should not access this data. Data encryption is thus required to secure the data. We consider gpg (GNU Privacy Guard) as part of our benchmark suite. We sign and encrypt the same 1.2 GB archive as for the compression benchmark.

Rich media applications. As mentioned before, the data stored online is becoming more and more rich media, including audio (e.g., iTunes, MySpace, Spotify), images (e.g., flickr), video (e.g., YouTube), as well as virtual reality (e.g., online games). We include three benchmarks to cover rich media applications, namely blender, bodytrack and x264. The blender benchmark is a 3D graphics rendering application for creating 3D games, animated film and visual effects. We render 40 frames from a 3D scene including objects, and shadow, lightning and mirroring effects. The bodytrack benchmark is a computer vision application that tracks a human body with multiple cameras through an image sequence. As input data we consider 200 frames from 4 cameras with 4,000 particles in 5 annealing layers (input data set of 477 MB). The x264 benchmark is an application for encoding video streams in H.264 format. Its input is a 1.5 GB video file.

Classical business logic. Next to these emerging workloads, classical business logic will remain to be an important workload. We include PseudoSPECjbb2005, a modified version of SPECjbb2005 that executes a fixed amount of work rather than for a fixed amount of time. SPECjbb models the middle tier (the business logic) of a three-tier business system containing a number of warehouses that serve a number of districts. There are a set of operations that customers can initiate, such as placing orders or requesting the status of an existing order. PseudoSPECjbb, in our setup, processes 4 M operations in total.

Both multi-threaded as single-threaded workloads. As mentioned in Table 3.1, we gathered these benchmarks from various sources. Some benchmarks come from existing benchmark suites (PARSEC [10], MineBench [56], BioPerf [4]), while others were derived from real-life applications (Apache lucene, blender, GNU gpg, GNU tarz). Half the benchmarks are multi-threaded workloads (blender, bodytrack, kmeans, specjbb, x264); the others are single-threaded (hmmer, eclat, gpg, lucene, tarz). The inputs for these workloads were chosen such that the run time on a dual-processor dual-core AMD Opteron 2212 machine is on the order of minutes, see also Table 3.1. We simulate these workloads to completion.

Workload data set sizes. All the workloads run on data sets with hundreds of MBs or on the order of GBs of data. Although the data sets may be even bigger in real setups, we believe this is a reasonable assumption for our purpose, because these data sets do not fit in the processor's caches anyway. Hence, simulating even larger data sets is unlikely to change the overall conclusions. We simulate these workloads for minutes of real time, see also Table 3.1, or hundreds of billions of instructions, which is unusual for architecture simulation studies.

Category	Benchmark	Source	Description	Run time
data compression	tarz	GNU	Create an archive and compress the files	1m10s
data mining	kmeans	MineBench	Mean-based data clustering	1m50s
	eclat	MineBench	Association rule mining to find interesting relationships in large data sets	1m56s
	hmmer	BioPerf	Compares sequence alignments against hidden Markov models	3m30s
data indexing	lucene	Apache	Apache text search indexer library written in Java	1m59s
data security	gpg	GNU	Sign and encrypt files	1m30s
rich media	blender	Blender Foundation	3D graphics rendering for creating 3D games, animated film or visual effects	2m15s
	bodytrack	PARSEC	Body tracking using multiple cameras	1m38s
	x264	PARSEC	Encoding video streams in H.264 format	1m15s
business	SPECjbb2005	SPEC	Middle-tier of server-side Java performance	2m09s

Table 3.1: Our set of data-centric benchmarks: their category, source, description and run time on a dual-socket dual-core AMD Opteron 2212 machine.

3.3 Data center Modeling

Data center design is very much cost driven, and design decisions are driven by two key metrics, namely performance and cost [6]. Cost is not limited to hardware cost, but also includes power and cooling as well as data center infrastructure cost. A recently proposed metric for Internet-sector environments is performance divided by total cost of ownership (TCO) and quantifies the performance achieved per dollar [47]. We now describe how we quantify cost and performance in the following two subsections, respectively.

3.3.1 TCO modeling

We build on the work by Lim et al. [47] to quantify data center cost. A three-year depreciation cycle is assumed and cost models are provided for hardware cost, as well as power and cooling costs. Hardware cost includes the individual components (CPU, memory, disk, board, power and cooling supplies, etc.) per server. Power and cooling cost includes the power consumption of the various server and rack components. The cooling cost includes infrastructure cost for power delivery, infrastructure cost for cooling, and the electricity cost for cooling.

We consider three server types: a high-end server, a low-end embedded processor and a middle-of-the-road (commodity) server. Table 3.2 describes their configurations and their cost models. The high-end server that we simulate is modeled after the Intel Xeon X5570; we assume an eight-core machine running at 3 GHz with a fairly aggressive out-of-order processor core along with an aggressive memory hierarchy.[3] The low-end processor is a dual-core embedded processor running at 1.2 GHz with a modest core and memory hierarchy, and is modeled after the Intel Atom Z515 processor. The commodity system is somewhat in the middle of the road between the high-end and low-end systems. We assume 4 cores at 2 GHz and we model it after the Intel Core 2 Quad. The cost for each of the components is derived from a variety of sources.[4,5,6] We use these default costs for reporting a reasonable design point given today's technology. Note that we do account for the server Network Interface Card (NIC) cost as part of the 'board and management' cost. We do not account for the network itself; we basically assume that network cost is constant across different data center configurations. We believe this is a reasonable first-order approximation,

[3]The Intel Xeon X5570 implements 4 cores and 2 hardware threads per core

[4]http://ark.intel.com/Product.aspx?id=40740

[5]http://www.newegg.com/Product/Product.aspx?Item=N82E16813131358

[6]http://ark.intel.com/Product.aspx?id=40816&processor=Q8200S&spec-codes=SLG9T

Processor configuration			
	high-end	middle	low-end
frequency	3 GHz	2 GHz	1.2 GHz
#cores	8	4	2
OOO core	4-wide	3-wide	2-wide
ROB size (#insns)	160	90	40
mem latency (cycles)	120	80	40
private L1 caches	64 KB	32 KB	32 KB
L1 prefetching	yes	yes	no
private L2 caches	256 KB	NA	NA
shared LLC cache	8 MB	2 MB	1 MB
LLC prefetching	yes	no	no
branch predictor	4 KB, 14 b hist	2 KB, 10 b hist	1 KB, 8 b hist
Cost model			
	high-end	middle	low-end
CPU	1,386	213	45
board and mngmnt	330	145	50
memory	265	113	98
total hardware cost	1,981	471	193
CPU power (TDP)	95	65	1.4
server & rack power	300	100	22
cooling	300	100	22
total power (Watt)	600	200	44
power cost 3-year	2,680	894	197
total cost 3-year	4,662	1,365	390

Table 3.2: Processor configurations and their cost models (in Euro).

given that networking accounts for 8% of the total data center cost only [29]. Further, because cost depends on many sources and varies over time, we vary the relative cost ratios across platforms in order to understand cost sensitivity in Section 3.5. In other words, if server cost and/or network cost were to differ across data center configurations, this can be accounted for through these cost ratios.

We consider a default energy cost of 17 Eurocent per kWh, unless mentioned otherwise. This is a typical private tariff rate; industry tariff rate may be as low as 10 Eurocent per kWh and below, hence, we explore a range of electricity costs in the evaluation section of this chapter.

3.3.2 Performance modeling

We use HP Labs' COTSon simulation infrastructure [2] which uses AMD's SimNow [7] as its functional simulator to feed a trace of instructions into a timing model. COTSon can simulate full-system workloads, including the operating system, middleware (e.g., Java virtual machine) and the application stack. In this study, we use the COTSon-based simulator by Ryckbosch

et al. [68], which has been validated against real hardware and which runs at a simulation speed of 37 MIPS with sampling enabled. This high simulation speed enables us to run the data-centric workloads on sufficiently large datasets for minutes of real time. The sampling strategy assumed is periodic sampling: we consider 100 K instruction sampling units every 100 M instructions and 1 M instructions prior to each sampling unit for warming the caches and predictors.

We quantify performance as throughput or the number of jobs that can be completed per unit of time. Because the workloads that we consider are supposed to run as background processes in the cloud — these workloads are non-interactive with the end users — we believe throughput is the right performance metric. For each platform we compute the best possible throughput that can be achieved. For the single-threaded benchmarks this means we run multiple copies of the same benchmark concurrently on the multi-core processor and we vary the number of copies (e.g., for the high-end server, from one copy up to eight copies), and we then report the best possible throughput that was achieved (for the same input set). For the multi-threaded benchmarks, we vary both the number of copies and the number of threads (e.g., on an 8-core system we consider 1 copy with 8 threads, 2 copies with 4 threads, etc.), and we report the best possible throughput.

3.4 Optimizing the data center

3.4.1 Which server type is optimal?

Figure 3.1 quantifies performance per TCO efficiency for the high-end, middle-of-the-road and low-end servers, normalized to the high-end server. Performance per TCO efficiency is defined as TCO divided by performance, or the reciprocal of performance per TCO. Hence, performance per TCO efficiency is a lower-is-better metric. The interesting observation from Figure 3.1 is that there is no single winner: there is no single server that yields the best performance per TCO across all the workloads. For most workloads, the low-end server results in the lowest performance per TCO efficiency, however, for a couple workloads, the high-end server is the most performance per TCO efficient system, see for example kmeans and x264.

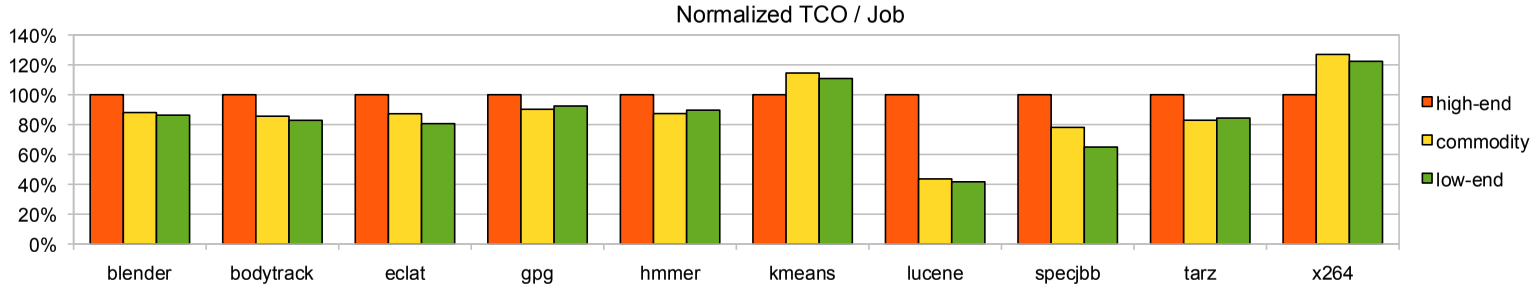

Figure 3.1: Normalized performance per TCO efficiency (lower is better) for the high-end, the middle-of-the-road and the low-end servers.

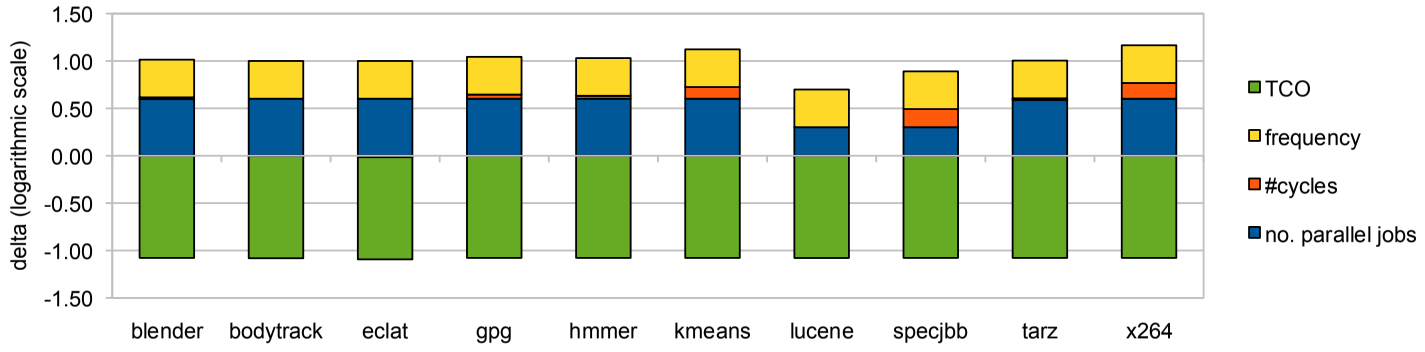

Figure 3.2: Performance per TCO stacks for quantifying the different factors; high-end versus low-end processors.

It is also interesting to note that for a couple workloads, namely gpg, hmmer and tarz, the middle-of-the-road server yields the best performance per TCO efficiency, albeit the difference with the low-end server is very small. The result that high-end and middle-of-the-road servers are more cost-efficient than low-end servers for some data-centric workloads is surprising and is in contrast to common wisdom and recently reported studies [1] [47] [65] which argued for lower-end servers to optimize cost-efficiency in the data center.

The reason is that these workloads are computation-intensive which makes the high-end and commodity servers yield a better performance per cost ratio, as we explain next. It must be noted that these conclusions hold true for our workloads, but more study is needed before we can generalize these results to a much broader range of data-centric workloads, and Internet-sector workloads in general.

3.4.2 Where does the benefit come from?

In order to get some insight as to why a particular server type is a winner for a particular workload, we break up the performance per TCO metric into its contributing components, using the following formula:

$$\textit{performance per TCO} = \frac{\textit{no. parallel jobs} \cdot \frac{\textit{freq}}{\textit{\#cycles}}}{TCO}. \tag{3.1}$$

The denominator quantifies cost for which we assume a 3-year depreciation cost cycle. The nominator quantifies throughput as the number of parallel jobs multiplied by the performance per job, or the reciprocal of the job's execution time; we measure throughput as the number of jobs that can be completed over a three-year time period. Figure 3.2 quantifies the contributing components when comparing the high-end versus the low-end server. The vertical axis is on a logarithmic scale. The contributing components are additive on a logarithmic scale, or multiplicative on a nominal scale. A negative component means that the component is a contributor in favor of the low-end server. In particular, TCO is always in favor of the low-end server because the TCO for the low-end server is about 12 times as low as for the high-end server. A positive component implies that the component is a contributor in favor of the high-end server. For example, frequency is a significant positive contributor for the high-end server: 3 GHz versus 1.2 GHz, a 2.5× improvement. Also, the number of parallel jobs is a significant contributor for the high-end server for most workloads. This means that the high-end server benefits from its ability to run multiple jobs in parallel, and hence achieve a higher throughput than the low-end server. Note that for some benchmarks, e.g., lucene and specjbb, this component

is only half as large as for the other benchmarks. This is due to the fact that 4 copies is the optimum for these benchmarks on the high-end servers versus 2 copies on the embedded server, whereas for the other benchmarks 8 copies is the optimum on the high-end server. Finally, the third positive contributor is the number of execution cycles; this means that the execution time in number of cycles is smaller on the high-end server compared to the low-end server. For most benchmarks, the number of execution cycles is roughly the same for the high-end and low-end servers, which implies that on the low-end server, the reduction in memory access time (in cycles) is compensated for by the increase in the number of cycles to do useful work (smaller processor width on the low-end server) and the increase in the number of branch mispredictions and cache misses (due to a smaller branch predictor and smaller caches on the low-end server). The number of execution cycles is a positive contributor for the high-end server for three benchmarks though, namely kmeans, specjbb and x264. In other words, the high-end server benefits significantly from the larger caches and branch predictor as well as the larger width compared to the low-end server for these workloads.

3.4.3 Does multi-threading help?

As mentioned before, half the workloads are multi-threaded and we optimize the data center for optimum throughput at the lowest possible cost. An interesting question is whether multi-threading helps if one aims for maximizing throughput. In other words, for a given workload for which there exists both a sequential and a parallel version, should we run multiple copies of the sequential version simultaneously, or are we better off running a single copy of the multi-threaded version? This is a non-trivial question for which an answer cannot be provided without detailed experimentation. On the one hand, parallel execution of sequential versions does not incur the overhead that is likely to be observed for the parallel version because of inter-thread communication and synchronization. On the other hand, multiple copies of sequential versions may incur conflict behavior in shared resources, e.g., the various sequential copies may incur conflict misses in the shared cache.

Table 3.3 summarizes the optimum workload configuration on each of the servers in terms of the number of instances of each workload and the number of threads per workload. For all of the multi-threaded workloads, except for specjbb, running multiple copies of the single-threaded workload version optimizes throughput. It is remarkable to see that multi-threading does not help in maximizing throughput for the data-centric workloads. Running multiple sequential versions yields higher throughput compared to running a single parallel version; co-running sequential

	high-end	middle	low-end
blender	c8t1	c4t1	c2t1
bodytrack	c8t1	c4t1	c2t1
eclat	c8t1	c4t1	c2t1
gpg	c8t1	c4t1	c2t1
hmmer	c8t1	c4t1	c2t1
kmeans	c8t1	c4t1	c2t1
lucene	c4t1	c4t1	c2t1
specjbb	c4t2	c2t2	c2t1
tarz	c8t1	c4t1	c2t1
x264	c8t1	c4t1	c2t1

Table 3.3: Workload configurations that maximize throughput on the high-end, commodity and low-end servers; 'cxty' means 'x' copies of the same workload with 'y' threads.

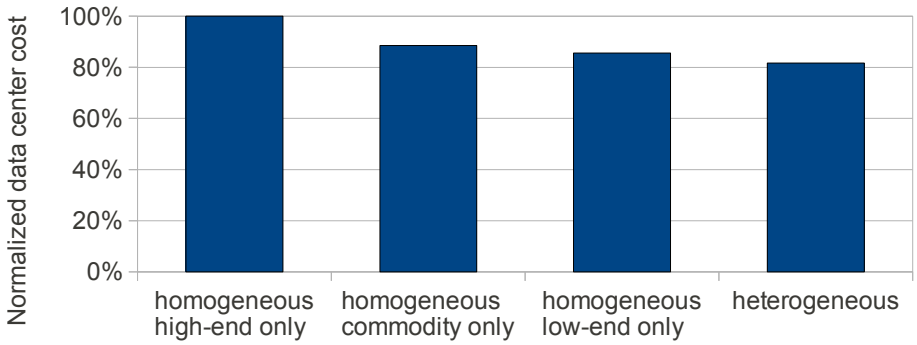

Figure 3.3: Normalized cost for iso-throughput homogeneous data centers with high-end, middle-of-the-road and low-end servers only, versus a heterogeneous data center.

versions does not incur significant conflict behavior in shared resources.

3.4.4 The case for a heterogeneous data center

The results shown above suggest that a heterogeneous data center in which a job is executed on the most cost-efficient server, may be beneficial. In order to quantify the potential of a heterogeneous data center for data-centric workloads, we consider four iso-throughput data center configurations. We consider three homogeneous data centers (with high-end servers only, middle-of-the-road servers only, and low-end servers only) as well as a heterogeneous data center. We assume the same workloads as before and we assume that all of these workloads are equally important — they all get the same weight. All of the data center configurations achieve the same throughput (for all of the workloads), hence, a data center with low-

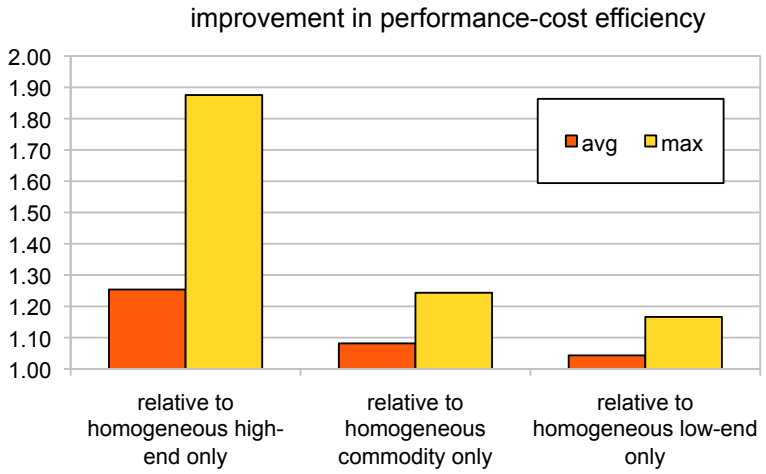

Figure 3.4: Cost reduction for a heterogeneous data center relative to homogeneous data center configurations across all possible two-benchmark workloads.

end servers needs to deploy more servers to achieve the same throughput as the homogeneous high-end server data center. The heterogeneous data center is configured such that it minimizes cost while achieving the same throughput as the homogeneous data centers.

Figure 3.3 quantifies data center cost normalized to the homogeneous high-end server data center. A homogeneous data center with commodity servers reduces cost by almost 12% and low-end servers reduce data center cost by 14%. A heterogeneous data center reduces cost by 18%. Clearly, optimizing the data center's architecture has a significant impact on cost. Even homogeneous data centers with commodity and low-end servers can reduce cost significantly. Heterogeneity reduces cost even further, although not by a large margin. However, this is very much tied to the workloads considered in this study. As shown in Figure 3.1, only two out of the ten workloads are run most efficiently on the high-end server. Hence, depending on the workloads, cost reduction may be larger or smaller.

In order to get a better view on the potential of heterogeneity as a function of its workload, we now consider a large variety of different workload mixes. The previous experiment assumed that all the workloads are equally important, simply because we do not have a way for determining the relative importance of these workloads in real data centers. We now consider a more diverse range of workload types: we consider all possible two-benchmark workload mixes and determine the potential benefit from

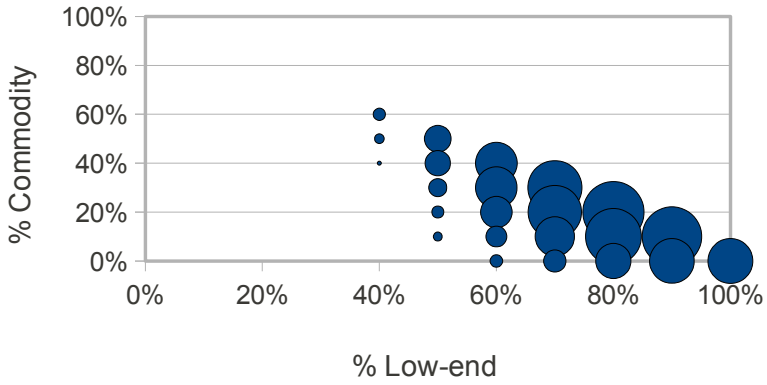

Figure 3.5: Configuration of the optimum heterogeneous data center: the fraction of low-end and commodity servers; the fraction of high-end servers equals one minus the fraction of low-end and commodity servers.

heterogeneity; this is to study how sensitive a heterogeneous data center is with respect to its workload. In other words, for each possible two-benchmark workload mix, we determine the cost reduction through heterogeneity relative to homogeneous data centers, see Figure 3.4. On average, a heterogeneous data center improves cost by 25%, 8% and 4%, and up to 88%, 24% and 17% relative to a homogeneous high-end, commodity and low-end server data center, respectively. (We consider the two-benchmark workload mixes for the remainder of this chapter.)

We now zoom in on the architecture of a heterogeneous data center. We therefore consider the workload mixes for which we observe a throughput benefit of at least 30% for heterogeneity compared to a homogeneous data center consisting of high-end servers only. Figure 3.5 plots the fraction of low-end and commodity servers in a heterogeneous data center; one minus these two fractions is the fraction of high-end servers. The size of the disks relate to the number of cases (workload mixes) for which we observe a particular configuration. We observe that the optimum heterogeneous data center typically consists of a relatively large fraction low-end servers and smaller fractions of commodity and high-end servers.

3.5 Sensitivity analyses

The results presented so far assumed the default parameters relating to data center cost mentioned in Section 3.3. Meaningful cost parameters are not easy to obtain because they are subject to a particular context, e.g., energy cost relates to where the data center is located, hardware purchase

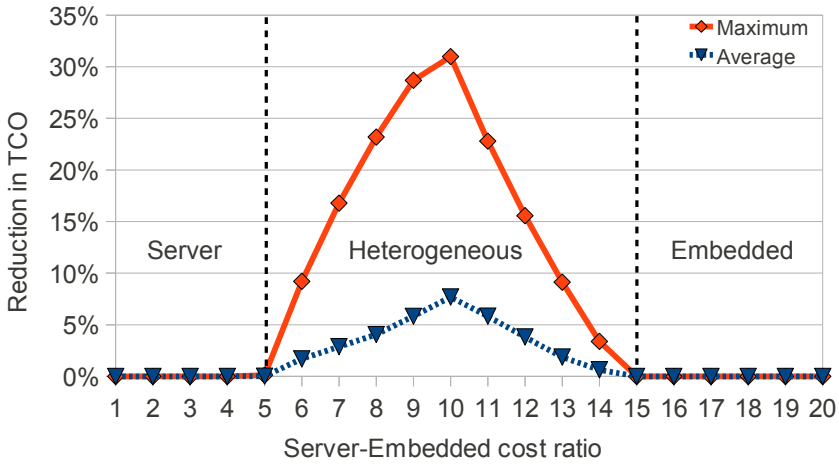

Figure 3.6: Cost reduction through heterogeneity as a function of the cost ratio between the high-end vs. low-end servers.

cost depends on the number of hardware items purchased, etc. In order to deal with the cost uncertainties, we therefore perform a sensitivity analysis with respect to the two main cost factors, hardware purchase cost and energy cost.

3.5.1 Varying the cost ratio

So far, we considered fixed costs for the various server types, as shown in Table 3.2. However, cost may vary depending on the number of servers that are bought — we assumed a fixed price per server. In addition, prices fluctuate over time. Hence, making a quantitative statement about which system is most performance-cost efficient at a given point in time, is subject to the cost ratios and thus it is not very informative. Instead, we also report the cost reduction through heterogeneity as a function of the cost ratio between the high-end and the low-end server, see Figure 3.6. The cost reduction reported here is the cost reduction over the best possible homogeneous data center. In case a high-end server is less than 5 times more expensive than a low-end server, then a high-end server is the clear winner, and there is no need for heterogeneity: a homogeneous high-end server data center optimizes the performance per dollar metric. In case a high-end server is more than 15 times more expensive that a low-end server, then a homogeneous data center with low-end servers is the optimal data center configuration. For cost ratios between 5× and 15×, performance per cost is optimized through heterogeneity. A 10× cost ratio yields the best possible benefit through heterogeneity, with an average reduction in cost of 8% and up to 31% for some workload mixes. As a point of reference, the cost

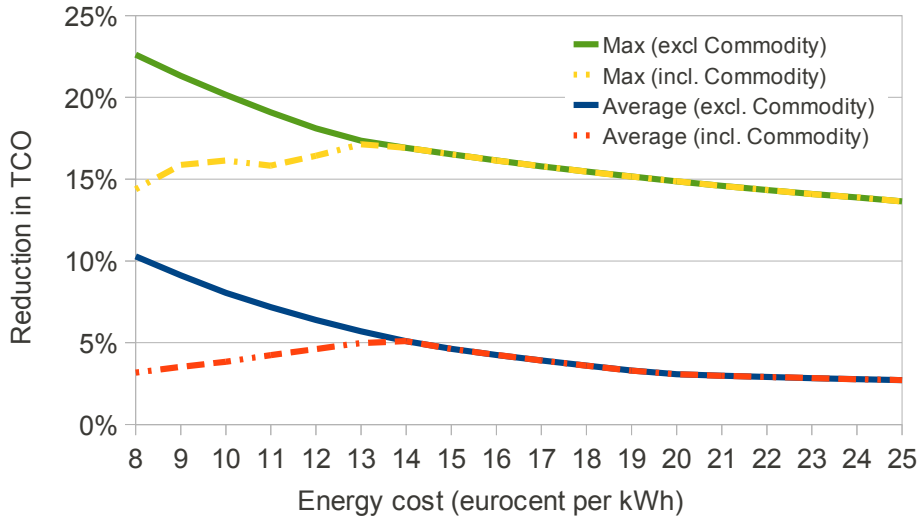

Figure 3.7: Cost reduction for a heterogeneous data center relative to the best possible homogeneous data center as a function of energy cost.

ratio between the high-end and low-end server assumed in the rest of this chapter equals 12, see Table 3.2.

3.5.2 Varying energy cost

Along the same line, energy cost is variable as well: it varies from one location to another, and it varies over time. Figure 3.7 quantifies the cost reduction of a heterogeneous data center relative to the best possible homogeneous data center as a function of energy cost. We report the average and maximum cost reduction through heterogeneity, and we consider two heterogeneous data center setups: one setup includes high-end, commodity and low-end servers, while the other includes high-end and low-end servers only (no commodity servers). The reason for considering both configurations is that the performance-cost efficiency is comparable for the commodity and low-end servers, as seen in Figure 3.1, which implies that heterogeneous data centers with high-end and low-end servers only would achieve most of the benefits from heterogeneity — including commodity servers does not add much benefit. This is indeed the case for the 17 Eurocent per kWh assumed so far.

The interesting observation from Figure 3.7 is that there is a cost benefit from heterogeneity across a broad range of energy prices. Second, when considering high-end and low-end servers only (i.e., 'excluding commodity' servers in Figure 3.7) for both the homogeneous and heterogeneous design points, the benefit from heterogeneity tends to be higher at lower energy costs. At lower energy costs, the performance argument outweighs

the cost argument, shifting the optimum towards high-end servers for a larger fraction of the workloads. At higher energy costs, the performance per cost metric drives the optimum design point towards low-end servers for most of the workloads, hence, the benefit from heterogeneity is decreasing. Finally, commodity servers fit a sweet spot at lower energy costs (see the 'including commodity' curves in Figure 3.7). Commodity servers have interesting performance-cost properties at low electricity costs — they yield good throughput at relatively low cost. Nevertheless, heterogeneity is still beneficial and can reduce the data center's TCO by up to 15%.

3.5.3 Discussion

The results discussed so far made the case for heterogeneous data centers. Significant cost reductions can be obtained compared to homogeneous data centers while achieving the same overall system throughput. The results also revealed that the extent to which cost is reduced is subject to various factors including the workloads, server cost ratio for different server types, energy cost, etc. Hence, in some cases, depending on the constraints, the benefit from heterogeneity may be limited. However, in a number of cases (for specific sets of workloads, server cost ratios and energy cost), heterogeneity may yield substantial cost benefits, which may translate into millions of dollars of cost savings in real data center environments.

3.6 Related Work

Prior work in architectural studies for warehouse-sized computers considered online interactive workloads for the most part. In particular, Lim et al. [47] consider four Internet-sector benchmarks, namely websearch (search a very large dataset within sub-seconds), webmail (interactive sessions of reading, composing and sending e-mails), YouTube (media servers servicing requests for video files), and mapreduce (series of map and reduce functions performed on key/value pairs in a distributed file system). These benchmarks are network-intensive (webmail), I/O-bound (YouTube) or exhibit mixed CPU and I/O activity (websearch and mapreduce). The data-centric benchmarks considered in this work are data-intensive and are primarily compute- as well as memory-intensive, and barely involve network and I/O activity. It is to be expected that cloud data centers will feature both types of workloads, interactive Internet-sector workloads as well as data-intensive background workloads. Lim et al. reach the conclusion that lower-end consumer platforms are more performance-cost efficient — leading to a 2× improvement relative to high-end servers. Low-end embedded servers have the potential to offer even more cost savings at the

same performance, but the choice of embedded platform is important. We conclude that heterogeneity with both high-end and low-end servers can yield substantial cost savings.

Andersen et al. [1] propose the Fast Array of Wimpy Nodes (FAWN) data center architecture with low-power embedded servers coupled with flash memory for random read I/O-intensive workloads. Vasudevan et al. [75] evaluate under what workloads the FAWN architecture performs well while considering a broad set of microbenchmarks ranging from I/O-bound workloads to CPU- and memory-intensive benchmarks. They conclude that low-end nodes are more energy-efficient than high-end CPUs, except for problems that cannot be parallelized or whose working set cannot be split to fit in the cache or memory available to the smaller nodes — wimpy cores are too low-end for these workloads. Whereas the FAWN project focuses on energy-efficiency, we focus on cost-efficiency, i.e., performance per TCO. While focusing on data-centric workloads, we reach the conclusion that both high-end and low-end CPUs can be cost-efficient, depending on the workload.

Reddi et al. [65] evaluate the Microsoft Bing web search engine on Intel Xeon and Atom processors. They conclude that this web search engine is more computationally demanding than traditional enterprise workloads such as file servers, mail servers, web servers, etc. Hence, they conclude that embedded mobile-space processors are beneficial in terms of their power efficiency, however, these processors would benefit from better performance to achieve better service-level agreements and quality-of-service.

Keys et al. [40] consider a broad set of workloads as well as different processor types, ranging from embedded, mobile, desktop to server, and they aim for determining energy-efficient building blocks for the data center. They conclude that high-end mobile processors have the right mix of power and performance. We, in contrast, aim for identifying the most cost-efficient processor type taking into account total cost of ownership (TCO), not energy-efficiency only. We conclude that a mix of high-end servers and low-end servers optimizes performance per TCO.

Nathuji et al. [57] study job scheduling mechanisms for optimizing power efficiency in heterogeneous data centers. The heterogeneous data centers considered by Nathuji et al. stem from upgrade cycles, in contrast to the heterogeneity 'by design' in this work. Also, Nathuji et al. consider high-end servers only and they do not include commodity and low-end servers as part of their design space.

Kumar et al. [44] propose heterogeneity to optimize power efficiency in multi-core processors. Whereas Kumar et al. focus on a single chip and power efficiency, our work considers a data center, considers total cost (in-

cluding hardware, power and cooling cost) and data-centric workloads.

Since the original publication of this work at the International Conference on Supercomputing in 2011, the research community has done more effort in studying contemporary data center workloads. Meisner et al. [52] study energy proportionality for one particular workload, namely web search. This workload requires responsiveness in the sub-second time scale at high request rates while performing significant computing over massive data sets per user request. They report the impact of using low-power modes on power usage and end-user latency. They conclude that there is still need for improving energy proportionality in shared caches and on-chip memory controllers.

Wong et al. [78] present an insightful survey of server energy proportionality and conclude that contemporary servers are particularly energy-inefficient at low utilization. They therefore present KnightShift, a server node design including a high-end and a low-end server sharing the disk — effectively bringing heterogeneity inside the node as opposed to heterogeneity at the cluster level. The idea behind KnightShift is to handle incoming requests by the low-end server (the Knight) at low activity and only engage the high-end server at high degrees of activity.

Ferdman et al. [22] focus their work on emerging scale-out workloads running in a cloud environment. They bundle several cloud workloads into a new benchmark suite called CloudSuite. They use performance counters to analyze the micro-architectural behavior of these workloads and conclude that there is a mismatch between the workload needs and modern processors, particularly in the organization of instruction and data memory systems and the processor core micro-architecture.

Lotfi-Kamran et al. [48] consider emerging applications in scale-out data centers. They show that performance of scale-out workloads that operate on large datasets is maximized through scale-out processors, representing microarchitectures having a modestly-sized last-level cache that captures the instruction footprint at the lowest possible access latency, instead of microarchitectures that waste silicon on excessively large caches.

Grot et al. [28] analyze several workloads from the CloudSuite benchmark suite as well as the SPECweb2009 workload by using a combination of analytical models and full-system simulation to estimate the performance, area and power usage. The authors consider five server chip architectures with varying core count, last-level cache size, clock frequency and power usage. Their results show that small-chip designs are significantly less expensive and more energy-efficient than conventional processors on a per-unit basis, however more expensive at the data-center level. The reason for this is the small chip's limited computing power, requiring many more units to achieve the same throughput. The authors conclude

that scale-out processors, as proposed by Lotfi-Kamran et al., provide a good balance between energy efficiency and performance.

Mars et al. [50] investigate microarchitectural heterogeneity in the data center, where heterogeneity comes from using several generations of hardware at the same time. The hardware is assumed to be homogeneous, however, successive generations of hardware have slightly different characteristics, leading to heterogeneity. The authors conclude that applications that are sensitive to heterogeneity can have a performance improvement of up to 70% when executed on the hardware platform that is most suitable.

3.7 Conclusion

Data explosion and diversity in the Internet drives the emergence of a new set of data-centric workloads to manage, manipulate, mine, index, compress, encrypt, etc. huge amounts of data. In addition, the data is increasingly rich media, and includes images, audio and video, in addition to text. Given that the data centers hosting the online data and running these data-centric workloads are very much cost driven, it is important to understand how this emerging class of applications affects some of the design decisions in the data center.

Through the architectural simulation of minutes of run time of a set of data-centric workloads on a validated full-system x86 simulator, we derived the insight that high-end servers are more performance-cost efficient compared to commodity and low-end embedded servers for some workloads; for others, the low-end server or the commodity server is more performance-cost efficient. This suggests heterogeneous data centers as the optimum data center configuration. We conclude that the benefit from heterogeneity is very much workload and server-cost and electricity-cost dependent, and, for a specific setup, we report improvements up to 88%, 24% and 17% over a homogeneous high-end, commodity and low-end server data center, respectively. We also identify the sweet spot for heterogeneity as a function of high-end versus low-end server cost, and we provide the insight that the benefit from heterogeneity increases at lower energy costs.

Chapter 4

Web 2.0 Workload Characterization

In the previous chapter we focused on data-centric applications. Another important type of Web applications are social network applications. Designing data centers for Web 2.0 social networking applications is a major challenge because of the large number of users, the large scale of the data centers, the distributed application base, and the cost sensitivity of a data center facility. Hence, optimizing the data center for performance per dollar is far from trivial. Because social network applications are highly interactive, we focus on how data center design decisions affect user-perceived performance in terms of response times.

4.1 Introduction

Internet usage has grown by 566% over the past twelve years worldwide according to a recent study by Internet World Stats.[1] This fast increase is due to various novel Internet services that are being offered, along with ubiquitous Internet access possibilities through various devices including mobile devices such as smartphones, tablets and netbooks. Online social networking in particular has been booming over the past few years, and has been attracting an increasing number of customers. Facebook, for example, has more than 1 billion active users as of November 2012, and 50% of these users log on to Facebook at least once a day.[2] Twitter has more than 500 million users and generates more than 100 million tweet messages per day as of July 2011.[3] LinkedIn has more than 135 million profession-

[1] http://internetworldstats.com/stats.htm

[2] http://newsroom.fb.com/

[3] http://semiocast.com/publications/2012_07_30_Twitter_reaches_half_a_billion_accounts_140m_in_the_US

als around the world as of November 2011.[4] Netlog, a social networking site where users can keep in touch with and extend their social network, is currently available in 40 languages and has more than 94 million users throughout Europe as of January 2012.[5] Clearly, social networking communities have become an important part of digital life.

Designing the servers and data centers to support social networking is challenging, for a number of reasons. As mentioned above, social networks have millions of users, which requires distributed applications running in large data centers [5]. The ensemble of servers is often referred to as a warehouse-scale computer [6] and scaling out to this large a scale clearly is a major design challenge. Because of their scale, data centers are very much cost driven — optimizing the cost per server even by only a couple tens of dollars results in substantial cost savings and proportional increases in profit. There are various factors affecting the cost of a data center, such as the hardware infrastructure (servers, racks and switches), power and cooling infrastructure, operating expenditure, and real estate. Hence, data centers are very cost-sensitive and need to be optimized for the ensemble. As a result, operators drive their data center design decisions towards a sweet spot that optimizes performance per dollar.

A key question when installing a new data center obviously is which new hardware infrastructure, i.e., which servers, to buy. This is a non-trivial question given the many constraints. On the one hand, the hardware should be a good fit for the workloads that are going to run in the data center. The workloads themselves could be very diverse — some workloads are interactive, others are batch-style workloads and thus throughput-sensitive and not latency-critical; some workloads are memory-intensive while others are primarily compute-intensive or I/O-intensive. Hence, some compromise middle-of-the-road architecture may need to be chosen to satisfy the opposing demands; alternatively, one may opt for a heterogeneous system where different workloads run on different types of hardware. Further, one needs to anticipate what new workloads might emerge in the coming years, and how existing workloads are likely to evolve over time. On the other hand, given how cost-sensitive a data center is, it is of utmost importance that the correct hardware is purchased for the correct task. High-end hardware is expensive and consumes significant amounts of power, which leads to a substantial total cost of ownership. This may be the correct choice if the workloads need this high level of performance. If not, less expensive and less power-hungry hardware may be a much better choice.

It is exactly this purchasing question that motivated this work: Can we come up with a way of guiding service operators and owners of data cen-

[4] http://press.linkedin.com/about
[5] http://en.netlog.com/go/about

ters to what hardware to purchase for a given workload? Although this might be a simple question to answer when considering a single workload that runs on a single server, answering this question is quite complicated when it comes to a Web 2.0 social networking workload. A social networking workload consists of multiple services that run on multiple servers in a distributed way in a data center, e.g., Web servers, database servers, memcached servers, etc. The fundamental difficulty that a Web 2.0 workload imposes is that the performance of the ensemble can only be measured by modeling and evaluating the ensemble, because of the complex interplay between the various servers and services. In other words, performance as perceived by the end-user, i.e., the response times observed by the end user, is a result of the performance of the individual servers as well as the overall interaction among the servers. Put differently, optimizing the performance of an individual server may not necessarily be beneficial for the ensemble and may not necessarily have impact on end-user experience, nor may it have impact on the total cost of ownership.

In this chapter, we present a case study in which we characterize a real-life Web 2.0 workload and evaluate hardware and software design choices. We sample the Web 2.0 workload both in space and in time to obtain a reduced workload that can be replayed, driven by real input data. The reduced workload captures the important services (and their interactions) and allows for evaluating how hardware choices affect end-user experience.

We consider Netlog's commercially used Web 2.0 social networking workload, and we evaluate how hardware design choices such as number of cores, CPU clock frequency, hard-disk drive (HDD) versus solid-state drive (SSD), etc. affect overall end-user perceived performance. We conclude that the number of cores per node is not important for the Web servers in our workload, hence the hardware choice should be driven by cost per core; further, we find that the end-user response time is inversely proportional to Web server CPU frequency. SSDs reduce the longest response times by around 30% over HDDs in the database servers, which may or may not be justifiable given the significantly higher cost for SSD compared to HDD. Finally, the memcached servers show low levels of CPU utilization while being memory-bound, hence the hardware choice should be driven by the cost of integrating more main memory in the server.

We believe that this approach is not only useful to service providers and data center owners, but also to architects, system builders, and integrators to understand Web 2.0 workloads and how hardware choices affect user-perceived performance, server throughput, and utilization. Further, software developers and data center system administrators may find the approach useful to identify and solve performance bottlenecks in the software and experiment with alternative software implementations. To

demonstrate the potential usage of the characterization, we present two use cases illustrating how it can be leveraged for guiding hardware purchasing decisions and software optimizations.

This chapter is organized as follows. We first describe the Web 2.0 workload used in this study (Section 4.2). We then set the goals for this chapter (Section 4.3) and describe our methodology in more detail (Section 4.4). We detail our experimental setup (Section 4.5) and then present our results (Section 4.6). We focus on two important use cases for this work (Section 4.7). Finally, we discuss related work (Section 4.8) and conclude (Section 4.9).

4.2 Netlog Web 2.0 Workload

As mentioned in the introduction, we use Netlog's software infrastructure as a representative Web 2.0 workload. Netlog hosts a social networking site that is targeted at bringing people together. As of January 2012, Netlog is currently available in 40 languages and has more than 94 million members throughout Europe. According to ComScore, in January 2012 Netlog is the pageview market leader in Belgium, Italy, Austria, Switzerland, Romania and Turkey; and it is the second market leader in the Netherlands, Germany, France and Portugal.[6] Netlog has around 100 million viewers per month, leading to over two billion pageviews per month. Netlog users can chat with other friends, share pictures, write blog entries, watch movies and listen to music.

Netlog's architecture is illustrated in Figure 4.1. A load balancer distributes the incoming requests among the Web servers. The Web servers process the requests and assemble a response by fetching recently accessed data from the memcached servers. If the requested data is not present in one of the memcached servers, the Web server communicates with one of the database servers. There is one global database that holds general information with user data (like nickname and passwords). All other user data is spread among multiple database servers using a technique called 'sharding'.[7] Each of the servers run on a physical machine. The relative fraction of servers is as follows: 54% of Netlog's servers are Web servers, 16% are memcached servers and 30% are database servers. The Netlog data center hosts more than 1,500 servers.

Netlog's data center is partitioned among the languages that it sup-

[6] http://www.comscore.com/

[7] Sharding is a horizontal partitioning database design principle whereby rows of a database table are held separately, rather than splitting by columns. Each partition forms part of a shard, which may in turn be located on a separate database server or physical location.

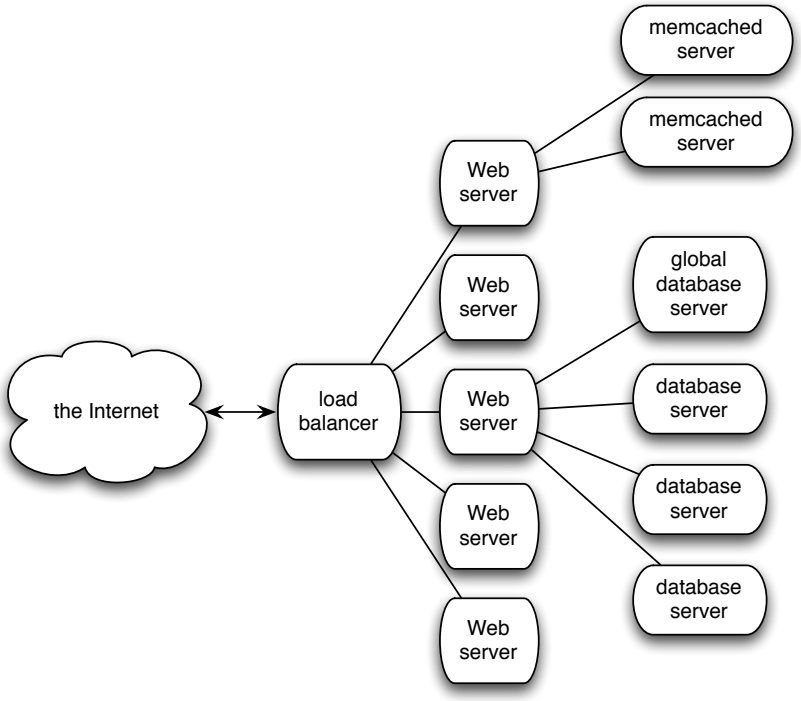

Figure 4.1: Netlog's architecture.

ports, i.e., servers are devoted to one particular language. The largest language is Dutch, followed by German, Italian, Arabic, English, and others. Interestingly, usage patterns are similar across languages, hence, the same relative occurrence of Web, caching and database servers is maintained across all the languages.

In terms of software, the Web servers run the Apache[8] HTTP server; the caching servers run Memcached[9]; and the database servers run MySQL[10]. For more information, please refer to Section 4.5.

4.3 Case Study Goals

Before describing our case study in great detail, we first need to set out its goals. First, we want to be able to characterize and evaluate end-user perceived performance of a Web 2.0 system. This implies that a representative part of the workload needs to be duplicated in the experimental environment which enables evaluating overall end-to-end performance. This

[8]http://httpd.apache.org/
[9]http://memcached.org/
[10]http://www.mysql.com/

in turn implies that a set of machines needs to be engaged with each machine running part of the workload — some run Web servers, some run database servers, others run memcached servers. Collectively, this set of machines runs the entire workload. This experimental environment, when supplied with real user requests, will act like a real data center running the real workload. This enables measuring user-perceived response times as well as server-side throughput and utilization.

Second, the experimental environment by itself will not provide useful measurement data. It also needs a method to feed real-life user requests into the experimental environment. In other words, real user requests need to be captured and recorded in a real data center and then need to be replayed in our experimental environment. This will enable us to measure how design choices in hardware and software affect user-perceived performance as well as server throughput and utilization.

Third, in addition to being able to faithfully replay real-life user requests, it is useful to be able to stress the setup through experiments in which user requests are submitted at a fixed rate. This allows for gaining insight into the system's limits and how the system will react in case of high loads. For example, it allows for learning about how user-perceived response time is affected by server load. Or, it allows for understanding the maximum allowable server load before seeing degradations in user response times.

Finally, we need the ability to run reproducible experiments, or in other words, we want to draw similar performance figures when running the same experiment multiple times. This allow us to measure how changes in system configuration parameters affect performance. In the end, we want to use the experimental environment and change both hardware and software settings to understand how hardware and software design choices affect user-perceived performance as well as server-level throughput and utilization. This not only enables service providers and data center owners to purchase, provision and configure their hardware and software, it also enables architects, system builders and integrators, software developers, etc. where to focus when optimizing overall system performance.

4.4 Methodology

Our methodology has a number of important features in order to make the experimental environment both efficient and effective for carrying out our case study.

- **Sampling in space.** It is obviously prohibitively costly to duplicate an entire Web 2.0 workload with possibly hundreds, if not thousands,

of servers in the experimental environment. We therefore sample the workload in space and we select a reduced but representative portion of the workload as the basis for the experimental framework. For the Netlog workload, we select one language out of the many languages that Netlog's workload supports; this language is representative for the other languages and for the Netlog workload at large. Sampling in space allows us to evaluate a commercial Web 2.0 workload with hundreds of servers in real operation with only 10 servers in our experimental environment.

- **Sampling in time.** Replaying a Web 2.0 workload using real-life user input, as we will describe next, can be very time-consuming, especially if one wants to replay multiple days of real-life operation in the data center. Moreover, in order to understand performance trends across hardware and software design changes, one may need to explore many configurations and hence run the workload multiple times. This may make the experimental setup impractical to use. Hence, we analyze the time-varying behavior of the workload and we identify representative phases in the execution, which we sample from, and which we can accurately extrapolate performance numbers to the entire workload. Sampling in time allows to analyze only a few hours of real time while being representative for a workload that runs for days.

- **Workload warm-up.** Sampling in time implies that we evaluate only a small fraction of the total run time. A potential pitfall with this approach is that system state might be very different when replaying under sampling than if one were to replay a workload for days of execution. This is referred to as the cold-start problem. In other words, the system needs to be warmed up when employing sampling in time so that the performance characteristics during the evaluation are representative for as if we were to run the entire workload. Our methodology uses a statistics-based approach to gauge whether the system is warmed up sufficiently.

- **Replaying empirical user request streams.** As mentioned before, we capture and replay real-life user requests. The user request file that we store on disk and that we use as input to the experimental environment contains sufficient information for faithfully replaying real-life users requests. In other words, the input served to the load balancer of the Netlog workload is identical under replay as when we captured it during real-life operation.

We now discuss the various steps of our methodology in more detail.

4.4.1 Sampling in space

As part of this study we duplicated Netlog's workload. Because it is infeasible to duplicate Netlog's entire workload, we chose to duplicate a small part only, namely the part associated with the Slovene language. This is feasible to do, and leads to a representative workload. Netlog organizes its servers such that there are a number of physical servers per language domain. Hence, by selecting a language domain and by only duplicating that language domain, we sample in space while being representative for the entire workload. The Slovene part is representative for Netlog's entire workload because it exhibits the same partitioning of servers as the rest of Netlog's workload. Also, we observe similar degree of activity and access behavior (access to profiles, photos, videos, etc.) for the Slovene language as for the other languages.

Duplicating the Slovene language part of Netlog's workload can be done with a reasonable number of servers. Our setup includes 6 Web servers, 1 memcached server and 2 database servers; this distribution across server types is identical to what is observed for the entire Netlog workload, across all the languages. Further, our setup includes the entire Slovene database and all of its records. The data present in our duplicate copy is anonymized. This is done through hashing while maintaining the length of the records.

4.4.2 Validating the setup

Duplicating a Web 2.0 workload is a significant effort and involves fine-tuning various software settings and configurations, which necessitates proper validation. We validated our experimental framework both functionally and with respect to behavior and timing. In particular, we automatically verified whether the file sizes returned by the duplicated workload match the file sizes observed in the real workload. The reason for doing so is that some of the Web pages returned by the Web 2.0 workload are composed semi-randomly, and hence its content may not be perfectly identical when requesting the same page multiple times. In our experimental environment, we found that 99.3% of the responses fall within a 5% error bound with respect to file size compared to the real workload environment, as shown in Figure 4.2.

4.4.3 Replaying user requests

An important aspect of the experimental environment is the replay of user requests. In order to do so, we collect user requests as observed at the load balancer. The information collected by the user input recorder consists of

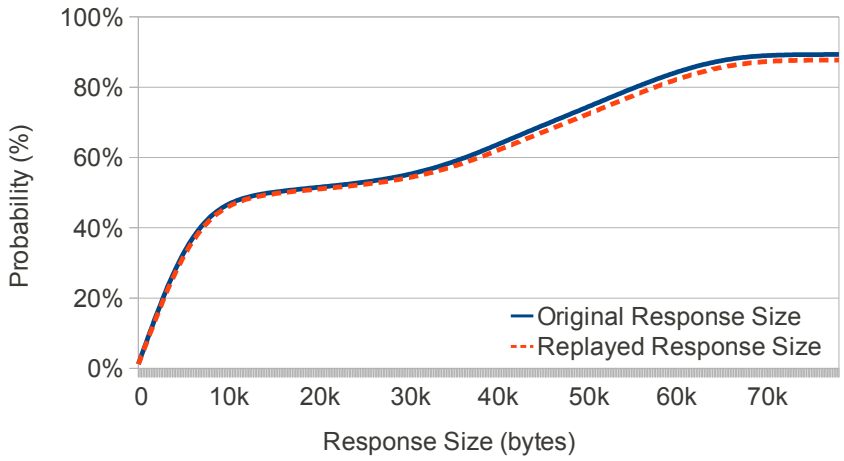

Figure 4.2: Distribution of response sizes when comparing real versus replayed requests.

the following items — recall that the data is anonymized:

- **Header information**. All HTTP header information is recorded so the same request can be reconstructed. This includes the requested URL, browser information, supported encoding formats, etc.
- **Timing information**. The date and time the request was submitted is recorded (at microsecond resolution). This allows for maintaining precise timing information when replaying the user request file. This is important to model bursty behavior in user requests.
- **User data**. The input recorder captures all POST data that is sent to the Web servers. Note that GET data is already captured as part of the URL in the header. All HTTP cookies are saved as well, and are used to do automated login.

The file that contains these user requests is fairly large and contains 24 GB of data per day on average. Our user input recorder uses `tcpdump` to log the network traffic to a file in `pcap` format.[11,12] `pcap` defines an API for capturing network traffic. On Linux/Unix systems, this is implemented in the `libpcap` library which most network tools like tcpdump, Wireshark, etc. implement. A limitation of `tcpdump/pcap` is that it may drop packets; however, packet loss rate was less than 0.002% for a 1 Gbps network in our setup.

The replayer reads the user request file and replays the requests one by one. This means that the replayer picks the first request, sends the request

[11] http://www.tcpdump.org/
[12] http://www.winpcap.org/ntar/draft/PCAP-DumpFileFormat.html

to the Web 2.0 workload at the time specified in the request file. It then picks the next request and sends it at its time, etc. The replayer does not wait for the response to come back to determine the next request; all the requests are available in the user request file.

Implementing the user request replayer is a challenge in itself. The reason is that the user request file is huge in size, and the requests need to be submitted to the workload at a fine time granularity. Reading the request file from disk, and submitting requests in real-time is too slow. On the other hand, it is impractical to store the entire request file in main memory. We therefore developed a two-thread replayer. The first thread reads the `pcap` file and fills in the requests in the request pool in memory. The second thread then reads from the request pool and submits the requests to the workload using `libcurl`, which is a client-side URL transfer library that supports sending requests using the HTTP protocol to a remote Web server.[13]

4.4.4 Sampling in time

We recorded four days (March 13–16, 2011) of user activity to the Slovene language domain of the Netlog workload. This was done by capturing all the user requests (and their timing) at the load balancer. Replaying these four days of activity in real time would require four days of experimentation time. Although this is doable if one were to evaluate a single design point, exploring trade-offs by varying hardware and/or software parameters, quickly leads to impractically long experimentation times. We therefore employ sampling in time to evaluate only parts of the workload activity while being representative for the entire workload.

Figure 4.3 shows traffic over a four day period in number of requests per second. Clearly, we observe cyclic behavior in which there is much more activity in the evening than during the day. Traffic increases steeply in the morning between 6am and 9am, and remains somewhat stable or increases more slowly between 9 am and 5pm. Once past 5pm, traffic increases steeply until 8pm. We observe a sharp decrease in the number of requests past 9pm. This traffic pattern suggests that sampling in time is a sensible idea, i.e., by picking samples that represent different traffic patterns, one can significantly reduce the load that needs to be replayed, which will lead to significant improvements in experimentation speed, while reproducing a representative workload.

We set ourselves a number of goals for how to sample in time. We want the samples to be representative in a number of ways: we want the samples to represent diverse traffic intensity as well as the sort of activity that

[13] http://curl.haxx.se/

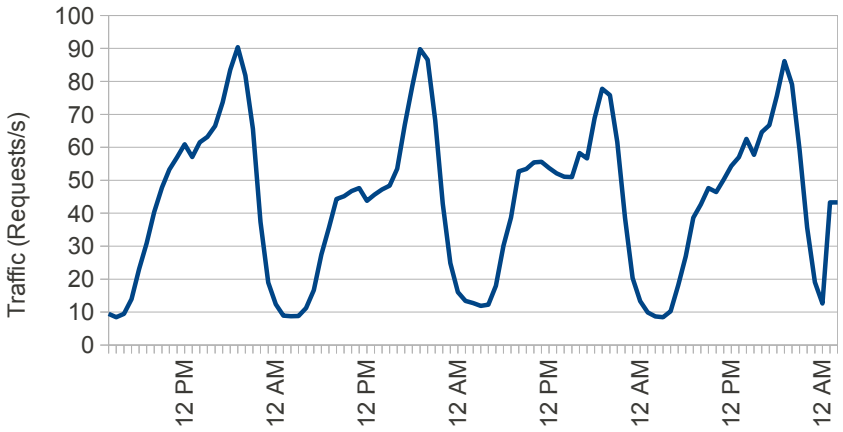

Figure 4.3: Netlog traffic profile for four days to the Slovene language domain.

the samples cover, i.e., as mentioned before, Netlog offers various sorts of services ranging from chatting to watching videos, etc., hence the samples should cover these different types of activity well. Further, we prefer having a few long representative samples over having many small samples. The reason is that small samples require more precise warmup of the system than longer samples in order to be accurate.

We therefore employ the following two-step sampling procedure. We first aim at finding a number of time periods with different traffic intensity. We employ k-means clustering as our classification method [36]. The input to the clustering algorithm is a time series representing the number of requests per minute. The clustering algorithm then aims at classifying this time series in a number of clusters N. It initially picks N cluster centroids in a random fashion, and assigns all data elements in the time series to its closest cluster. In the next iteration, the algorithm recomputes the cluster centroid, and subsequently reassigns all data elements to clusters. This iterative process is repeated until convergence, or until a maximum number of iterations is done. An important question is how many clusters N should one pick. We use the Bayesian Information Criterion (BIC) [71], which is a measure for how well the clustering matches the data. Using a maximum value of $N_{max} = 6$ — recall we aim for a limited number of samples — we obtain the result that $N = 3$ yields the optimum BIC score. Hence, we obtain three samples. These are shown in Figure 4.4. Intuitively, these three samples correspond to low-intensity, medium-intensity and high-intensity traffic, respectively.

The next question is how long the samples should be in these low, medium and high-intensity traffic regions. We therefore rely on our second requirement: we want the samples to cover diverse behavior in terms

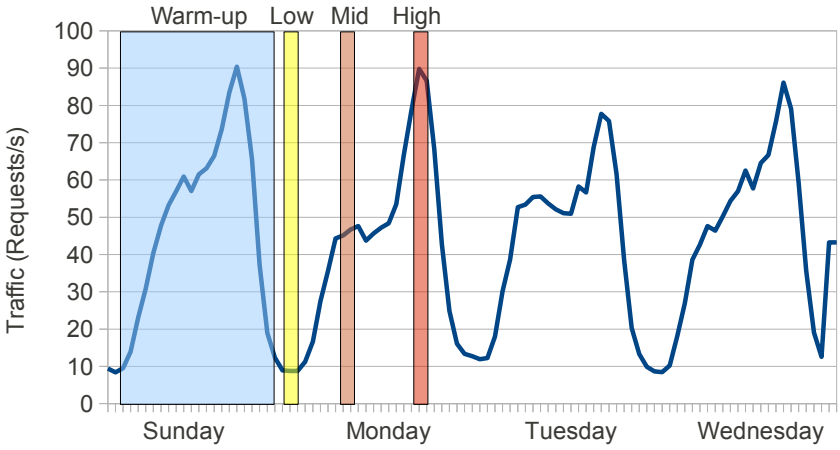

Figure 4.4: Identifying representative samples based on traffic intensity.

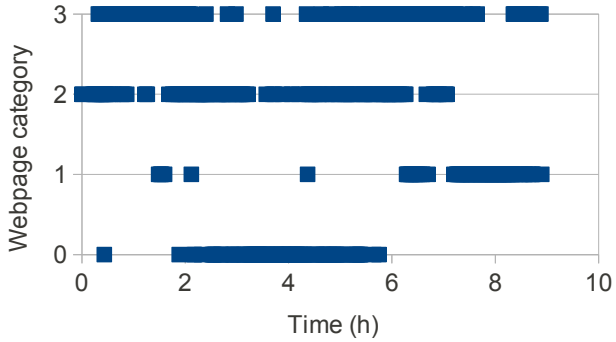

Figure 4.5: Traffic classified by its type.

of the type of traffic. We identify 30 major types of traffic including messages, photos, videos, friends, music, etc. This yields a 30-dimensional time series: each data element in the time series consists of 30 values, namely the number of requests per minute for each type of traffic. We then apply k-means clustering on this 30-dimensional time series which yields the optimum number of four clusters using the BIC score. These four clusters represent the predominant traffic rates observed at a given point in time. Figure 4.5 illustrates how the time series of ten hours of the second day is distributed across these four clusters. Interestingly, some traffic rates are more predominant during some periods of time, and traffic rate predominance varies fairly quickly. However, if we take a long enough snapshot, e.g., two hours, the sample contains all traffic rates. The end result for sampling in time, thus is that we pick three samples of two hours of activity from the low, medium and high-intensity regions.

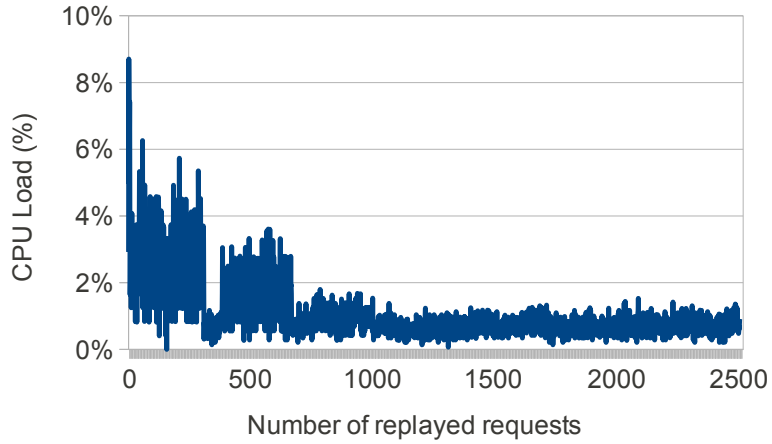

Figure 4.6: Quantifying PHP cache warmup behavior. Replay speed is set to a fixed rate of 10 requests/s.

4.4.5 Warmup

With sampling in time, an important issue is how to start from a warmed-up system state so that the performance numbers that we obtain from our experiments are representative for the real workload. Clearly, starting from a cold state is not going to be accurate because the performance of the workload will be very different from what one would observe in a real (and warmed-up) environment. Warmup of a Web 2.0 workload involves a number of issues. First, as mentioned before, the Web servers run PHP code, and hence they rely on an opcode cache that caches the bytecodes; the PHP engine does not need to interpret cached bytecodes again, and hence it achieves better performance. This implies that the performance of the PHP engine is relatively low initially, but then improves gradually as more and more code gets cached and optimized; this is obviously reflected in the Web server response times observed by the end user. In other words, in the context of this work, it is important that we measure the performance of the PHP engine in steady-state modus, in which it executes highly optimized code as opposed to interpreting the PHP code. As shown in Figure 4.6, the CPU load is higher when the PHP engine is first initialized. In this stage, the PHP engine still has to compile all PHP code. After 1,000 requests, most PHP pages are compiled and loaded into the cache, hence, we conclude that the PHP cache is warmed up in the order of a couple seconds.

Second, and more importantly, we also need to warm up the memcached and database servers. Initially, in a cold system, all the requests will go to the database server because the memcached server does not cache any data yet; further, the database server will need to read from disk to access the database. Hence, we will observe a significant fraction of time

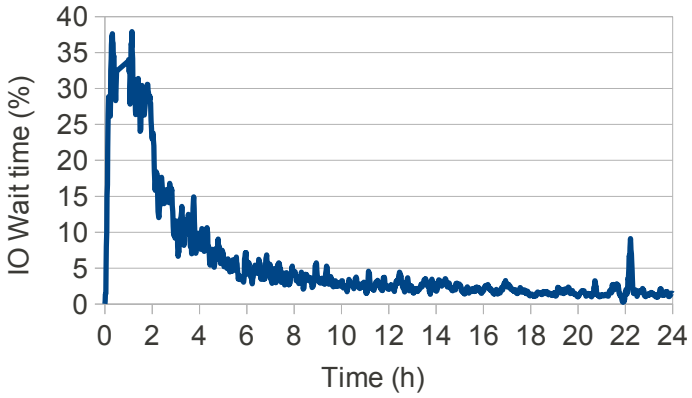

Figure 4.7: Quantifying how long one needs to warmup the database and memcached servers: I/O wait time on the database server is shown as a function of time when replaying the first day.

spent waiting for I/O both over the network and for accessing disks. Indeed, gigabytes of data need to be read in the database and transferred from the database servers to a memcached server. This requires a large number of user requests being sent to the system to warmup the database and memcached servers. Figure 4.7 illustrates the fraction I/O wait time on the database server starting from a cold state as a function of time. We observe that the fraction I/O wait time, which is proportional to how often one needs to access the database on disk and transfer data to the memcached server, decreases as a function of time. Although there is a steep decrease in I/O wait time in the first few hours, it takes close to an entire day before I/O wait time drops below a few percent which represents a fully warmed up system.

In order to get more confidence in this finding we employ the Kolmogorov-Smirnov statistical test to verify whether the system is sufficiently warmed up. The Kolmogorov-Smirnov test is a non-parametric test for the equality of continuous, one-dimensional probability distributions. It basically measures whether two distributions are equal or not; the exact form of the distribution is not important, hence it is labeled a non-parametric test. In this work, we compare the distribution of user response times starting from a cold versus a warmed-up system. This is done in steps of 5,000 user requests, see Figure 4.8. The P-value reported by the Kolmogorov-Smirnov test gives an estimate for how good the correspondence is between starting from no-warmup versus a fully warmed up system; the P-value is a higher-is-better metric. We observe that the P-value saturates after approximately six hours of warmup, and reaches its highest score after 18 to 20 hours of warmup. Based on these observations we decided to warm up our experimental system with one full day of load.

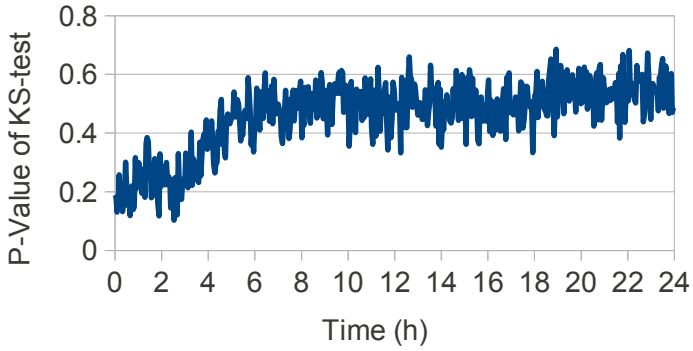

Figure 4.8: Using the Kolmogorov-Smirnov test to verify whether the system is sufficiently warmed up by comparing the distribution of response times under full versus no warmup.

Note that, in our experimental environment, it does not take a full day to actually warmup the entire system. During warmup, we quickly submit an entire day's user requests to the Netlog workload, as fast as possible. This takes approximately two hours in our setup. Once the system is warmed up, we then submit user requests for the sample of interest at the time stamps as stored in the user request file, as explained before.

4.5 Experimental Setup

As mentioned before, we duplicated the Slovene language domain of the Netlog workload to our experimental environment. Our infrastructure consists of 10 dual AMD Opteron 6168 servers, with each server having 24 cores in total or 12 cores per CPU. Each server has at least 64 GB of main memory, and is equipped with both a regular HDD (1 TB Seagate SATA 7200 rpm) as well as an SSD (128 GB ATP Velocity MII). We configure the machines as 6 Web servers, 1 memcached server, and 2 database servers. The tenth server is used to generate workload traffic and inject user requests to the system under test.

Our baseline configuration runs all the cores at 1.9 GHz. We provision the Web servers as well as the database servers with 64 GB of main memory. The memcached server is equipped with 128 GB of RAM. Further, we assume a HDD drive in each of the servers — we consider SSD in the database servers in one of the experiments.

Our infrastructure uses Ubuntu 10.04.[14] The Web server is configured

[14]http://www.ubuntu.com

with Apache 2.2[15] and runs PHP 5.2[16]. The database software used is a MySQL derivative, Percona 5.1. We use the standard Memcached 1.4.2[17] version as our caching mechanism.

4.6 Results and Discussion

Using our experimental environment, we now focus on gaining insights in how hardware trade-offs affect user-perceived performance (response times) for the end-to-end workload. We first consider user requests submitted at the rate as measured in the real-life workload, and we look at hardware trade-offs for the Web server, memcached server and database server, respectively. Subsequently, we consider fixed-rate experiments in order to stress the system.

4.6.1 Web server

We evaluate two hardware trade-offs for the Web server, namely CPU clock frequency and the number of cores per node. Figure 4.9 shows the distribution of the user response times while changing the Web server's CPU frequency in three steps: 1.9 GHz, 1.3 GHz and 800 MHz. The distribution of response times is skewed, i.e., there is a steep increase in the response time distribution around 0.04 seconds at 1.9GHz, and the distribution has a fairly long and heavy tail for longer response times. We observe similarly skewed distributions at lower CPU clock frequencies, yet the distributions shift towards the right with decreasing frequencies, i.e., user response time increases with lower clock frequencies. This is perhaps intuitive, as the CPU gets more work done per unit of time at higher clock frequencies. It is interesting to observe though that Web server clock frequency has a significant impact on user response times (even at low CPU loads, as we will see next). In conclusion, user response time is sensitive to Web server clock frequency. Hence, the Web server should have a sufficiently high clock frequency in order not to exceed particular bounds on user response time.

An important point of concern in provisioning servers for a Web 2.0 workload is to have sufficient leeway to accommodate bursty traffic behavior and sudden high peaks of load. For gauging the amount of leeway on a server, we use CPU load. If the CPU load is sufficiently low, the server can accommodate additional work. Figure 4.10 quantifies Web server CPU load as a function of clock frequency. Clearly, CPU load increases with lower CPU frequencies.

[15] http://www.apache.org/
[16] http://www.php.net
[17] http://www.memcached.org/

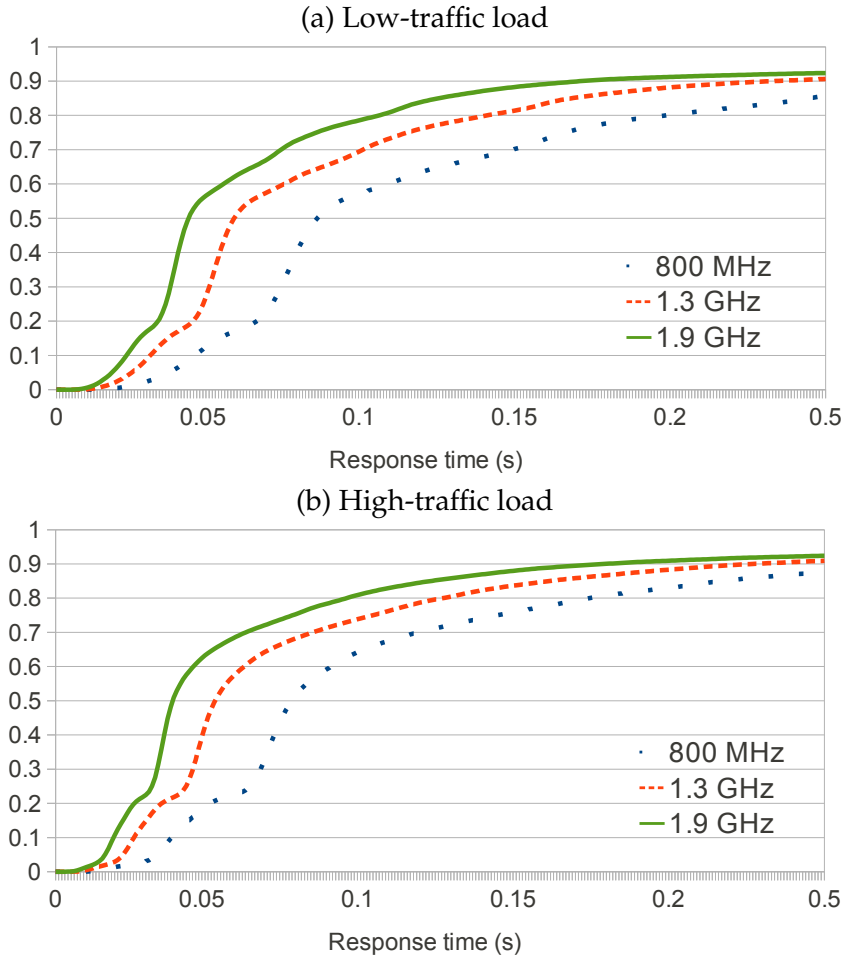

Figure 4.9: Cumulative distribution of user response times while changing the Web server's CPU frequency under (a) low-traffic load and (b) high-traffic load.

As alluded to before, the response time distribution has a fairly long and heavy tail. A heavy-tailed response time distribution is a significant issue in Web 2.0 workloads because it implies that some users are experiencing an unusually long response time. Given the large number of concurrent users of Web 2.0 workloads, and although the number is small in terms of percentages, still a significant number of users will be experiencing very long response times. Very long and unpredictable response times quickly irritate end users, which may have a significant impact on company revenue if users sign off because of the slow response times. Because of this, companies such as Google and others heavily focus on the 99% percentile of the user response times when optimizing overall system performance. Figure 4.11 shows the percentile response times as a function of Web server

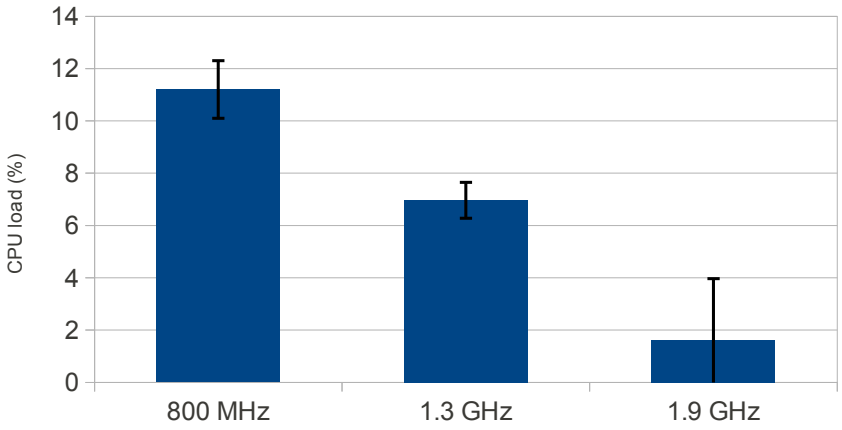

Figure 4.10: Web server CPU load as a function of CPU clock frequency for the high-traffic load scenario.

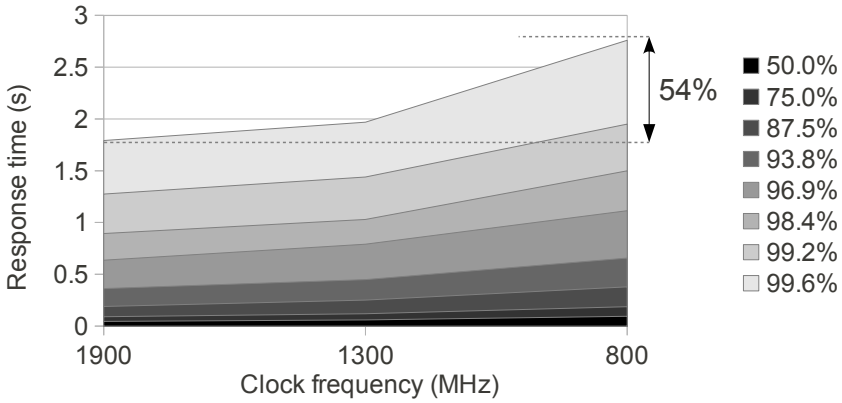

Figure 4.11: Percentile response times as a function of Web server CPU clock frequency.

CPU clock frequency. For example, this graph shows that, see the top left, 99.6% of the response times are below 1.8 seconds at 1.9 GHz. The 99.6% percentile goes up to 2.8 seconds at 800 MHz, see the top right. The interesting observation is that long-latency response times increase sub-linearly with decreasing Web server clock frequency. The 99.6% percentile response time increases by 54% only, while decreasing clock frequency from 1.9 GHz to 800 MHz, or increasing cycle time by 138% from 0.53ns to 1.25ns.

The second hardware trade-off that we study relates to the number of cores per node one should have for the Web server. The reason why this is an interesting trade-off is that systems with more sockets per node are more expensive, i.e., a four-socket system is typically more than twice as expensive as a two-socket system. Similarly, the number of cores per CPU

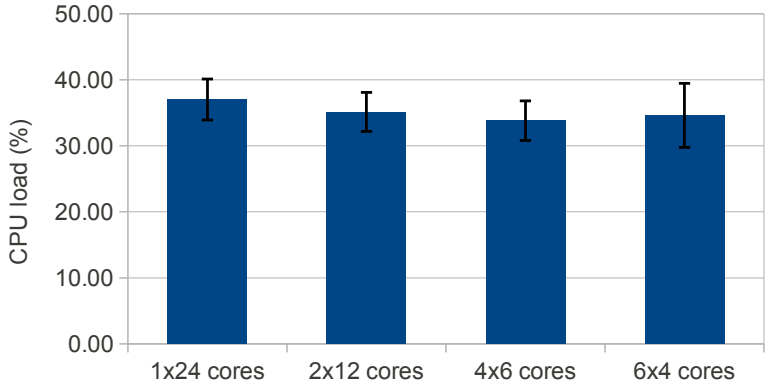

Figure 4.12: Web server CPU load as a function of the number of nodes and cores per node: $m \times n$ means m nodes and n cores per node.

also directly relates to cost. Figure 4.12 quantifies Web server CPU load as a function of the number of nodes and the number of cores per node. (Recall that CPU load is a good proxy for user response time as observed.) We vary from one Web server node with 24 cores enabled, to 2 nodes and 12 cores each, to 4 nodes and 6 cores each, to 6 nodes and 4 cores each. Clearly, CPU load (and response time) is not affected much by node and core count (as long as the total number of cores is constant). This suggests that the Web server is a workload that scales well with core count, even across nodes. In conclusion, when purchasing Web server hardware, although total core count is important, core count per node is not. This is an important insight to take into account when determining how many servers to buy with how many cores each. Determining the best buy (number of servers and number of cores per server) depends on many factors such as performance, power cost, real estate cost, reliability, availability, etc., however, this case study shows that the number of cores per server is a parameter one can tweak to optimize Web server performance per dollar.

4.6.2 Database server

As mentioned before, the database servers generate substantial disk I/O activity. We therefore focus on a hardware trade-off that involves HDDs versus SSDs in the database servers. We also vary CPU clock frequency. Figure 4.13 quantifies the percentile response times for the four hardware design points that result from changing clock frequency and hard drives. We observe that, while short response times are not greatly affected by replacing the HDD with an SSD, the 99.6% percentile response time decreases by 30% when trading an HDD for an SSD. Although this is a significant reduction in the longest response times observed, it may not justify the sig-

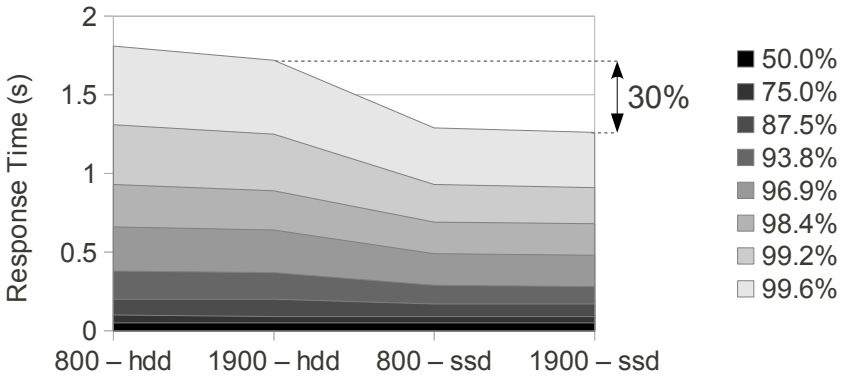

Figure 4.13: Trading off HDD versus SSD and CPU clock frequency for the database servers.

nificantly higher cost of SSD versus HDD.

4.6.3 Memcached server

The memcached server has a very low typical CPU load, and is primarily memory and network-bound. The average CPU load for the memcached server is typically below 5% when stressed with 6 Web servers. Figure 4.14 shows CPU time versus network time for a memcached experiment in which we generate memcached GET requests of varying size, more specifically, the responses of the GET requests are of varying size. This clearly shows that memcached performance is mainly determined by the network. Hence, CPU performance for the memcached servers is not critical, and one could for example deploy relatively inexpensive servers. It is important for the memcached servers to have sufficient amount of memory though.

4.6.4 Fixed-rate experiments

The experiments done so far involved replaying the user requests as recorded in the real-life workload, i.e., the requests are submitted at a rate determined by the users. We now consider experiments in which we submit requests at a fixed rate. The reason for doing so is to stress the system under high levels of request rates in order to understand how CPU load and user response times are affected by the load imposed on the entire workload. Figure 4.15 quantifies CPU load of the Web servers (left axis) and the average response time (right axis) as a function of the number of requests per second submitted to the system. Interestingly, the response time remains low and CPU load increases linearly with request rate, up until a request rate of 700 requests per second. Beyond 800 requests per

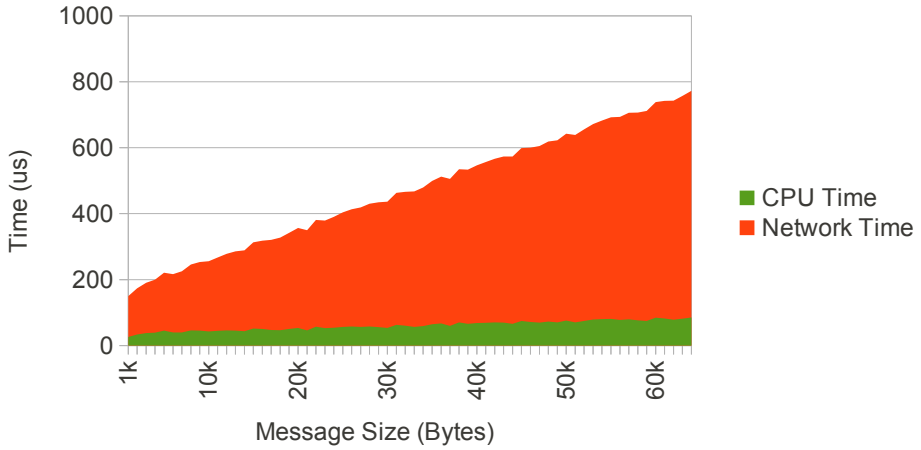

Figure 4.14: CPU time versus network time for memcached requests of different size.

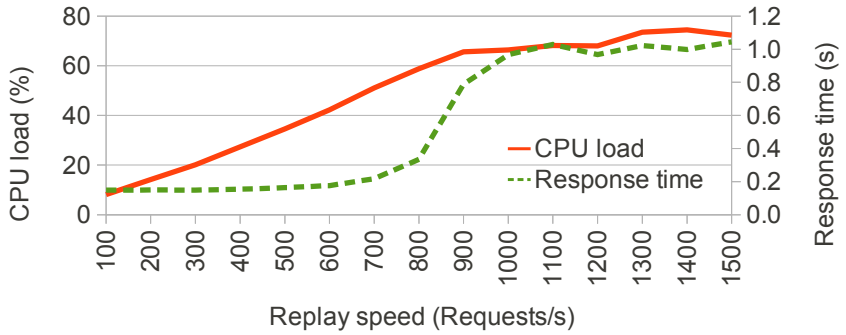

Figure 4.15: CPU load and average response time as a function of the number of requests per second under a fixed-rate experiment.

second, CPU load saturates around 65 to 75 percent, and response time increases substantially from 0.2 seconds to approximately 1 second. The reason why response time saturates beyond 1,000 requests per second is that requests get dropped once the Web server's CPU load gets too high as it runs out of resources and is unable to process all incoming requests.

4.7 Use Cases

The approach we have followed can be applied to numerous use cases. For example, data center owners and service providers can use the approach to guide purchasing decisions. Similarly, system architects, integrators and implementors can use the approach to gain insight in how fundamental design decisions of the data center architecture affect user perceived per-

formance. Finally, software architects and system administrators can use the approach to drive software design decisions, identify and address performance bottlenecks and evaluate alternative software implementations.

In this section, we present two use cases to illustrate the potential of the approach for making hardware and software design choices.

4.7.1 Hardware purchasing

The first use case that we present relates to hardware design choices, and more specifically to data center owners and service providers who wish to understand which hardware to purchase for a given workload. As mentioned earlier, this is a challenging question because of the many constraints one needs to deal with, ranging from purchasing cost, energy/power cost, cooling cost, performance, throughput, density, etc. In this use case, we look at two of the most important factors, namely performance and purchasing cost, for guiding the purchasing decisions. We also consider the implications on a third factor, namely power consumption.

Data center owners and service providers who wish to upgrade their hardware infrastructure face a challenging problem, and their decisions are mostly guided by experience and advice given by the hardware vendor(s). We now describe a scenario in which a hardware vendor would make a recommendation on which hardware to purchase. This scenario is hypothetical — we did not actually ask a hardware vendor for making a suggestion for a specific configuration for the given Web 2.0 workload. However, the suggested configuration is based on rules of thumb, and therefore we believe it is realistic. The hardware prices are based on real cost numbers of a large online hardware vendor. We now describe the suggested hardware configuration.

- **Web server**: It is well-known that Web servers are performance-hungry. Therefore, a hardware vendor might, for example, suggest a high-performance system with an Intel Xeon X3480 (3.06 GHz, 8 MB LLC Cache, 4 cores, Hyper-Threading), 8 GB RAM and a typical HDD. The price for this web server is $1,795.

- **Memcached server**: Because memory is an important factor in a memcached server, the vendor might suggest including more memory, leading to an Intel Xeon X3480, with 16 GB RAM and a typical HDD. The price for this system is $2,015.

- **Database server**: Finally, because the hard disk is often a bottleneck on a database server, a hardware vendor might suggest to replace the HDD with an SSD, leading to a system with an Intel Xeon X3480, 16 GB RAM and an SSD. The price for this database server is $2,915.

The total cost of this configuration, including 6 Web servers, 1 memcached server and 2 database servers — recall the 6-1-2 ratio of web, memcached and database servers in the Netlog configuration as described earlier — equals $18,615.

Now, given the insight obtained from this study as described in Section 4.6, we can make the following alternative recommendations for a hardware configuration.

- **Suggestion #1: Low-cost memcached and database server**

  As previously reported, a memcached server does not need a high-performance CPU. We therefore suggest a CPU of the same class as proposed by the hardware vendor, but at a lower clock frequency (e.g., Intel Xeon X3440 at 2.53 GHz). The same is true for the database server. On top of that we suggest not to consider an SSD, because of its high cost and relatively low performance gain over HDD for this particular workload.

  The lower price for the CPU makes the memcached and database server cost $1,445 each. The total price of our suggested configuration now equals $15,105. This means a purchasing cost reduction of 18.9%.

  Using the results presented in Figure 4.13, we conclude that, using this configuration, 50% of all requests will not experience any extra latency. For the other 50% of the requests, response times would increase from 11% for the 75% percentile to 39% for the 99.6% percentile. In summary, performance as perceived by the end user would be reduced by 9.1% on average. It is then up to the service provider to balance the purchase cost against the loss in performance for a small fraction of the user requests.

- **Suggestion #2: Low-frequency Web server**

  We can go one step further and use a CPU at lower clock frequency for the Web servers as well (Intel Xeon X3440 at 2.53 GHz). The price of the Web server is now $1,225. This could mean a total purchasing cost reduction of 37.2% over the hardware vendor suggested configuration.

  User requests would now observe a latency increase by 29% for the 50% percentile, and up to 56% for the 99.6% percentile. On average, end-user performance would be reduced by 36.9%. Again, it is up to the service provider to determine whether this loss in performance is worth the reduction in cost.

So far, when computing the cost reduction, we only focused on purchasing cost and we did not account for savings due to lower power consump-

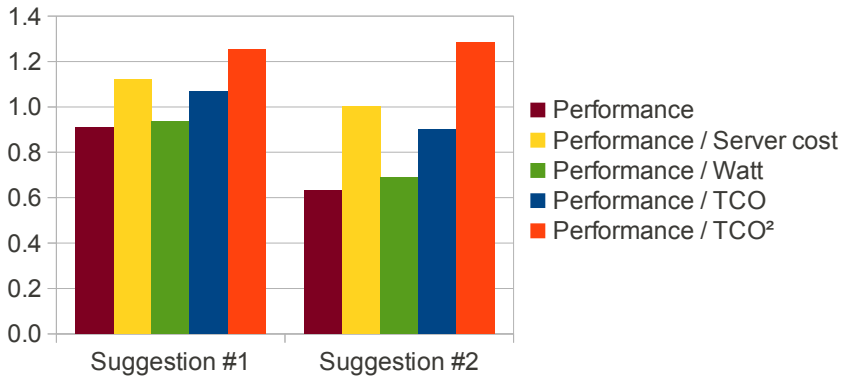

Figure 4.16: Several performance trade-offs for different hardware suggestions compared to the hardware vendor suggestion.

tion, leading to reduction in cost for powering and cooling the servers. Power consumption is a significant cost factor in today's servers and data centers [6], hence it should be taken into account when computing cost savings. In Figure 4.16, we show different metrics to help the service provider determine which platform should be chosen; the reason for considering multiple metrics is that different criteria might be appropriate for different scenarios. All metrics are higher-is-better metrics, and all values are normalized against the suggestion by the hardware vendor; a value greater than one thus is in favor of one of our suggestions.

- **Performance:** As mentioned before, raw performance drops by 9.1% for suggestion #1 and 36.9% for suggestion #2. This metric does not take any cost factor into account.
- **Performance per server cost:** Suggestion #1 reduces cost without dramatically reducing performance. When using server hardware costs in our metric, the benefit for suggestion #1 is 12.0%. The benefit for suggestion #2 is almost zero because of the extra decrease in performance, i.e., cost saving is offset by performance decrease.
- **Performance per Watt:** In the above case study, we considered two server configurations, one at 3.06 GHz and one at 2.53 GHz, which corresponds to a 17.3% reduction in clock frequency. Dynamic power consumption is proportional to clock frequency, so CPU dynamic power consumption will be lowered by 17.3% as well. The Intel X3480 has a Thermal Design Point (TDP) of 95 Watts and we assume other components (motherboard, disk, memory, etc.) to consume 100 Watts in total. The reduction in wattage is low compared to the performance decrease, resulting in a net decrease in performance per Watt for the two suggestions compared to the hardware vendor's

suggestion. However, this metric only considers power consumption and does not take electricity costs into account.

- **Performance per TCO:** As mentioned before, data center facilities and online services are cost-sensitive, and hence, a metric for the data center should include some notion of total cost of ownership (TCO). TCO includes server purchasing cost plus electricity cost for powering and cooling the servers. We assume electricity cost to be \$0.07/kWh and we assume a three-year depreciation cycle. For the cooling cost, we assume there is need for 1 Watt of cooling power for each Watt of consumed power. The three-year total cost of ownership (TCO) for 6 Web servers, 1 memcached server and 2 database servers consists of hardware cost, power cost and cooling cost; this makes \$24,887 for the hardware vendor's suggestion, \$21,201 for suggestion #1 and \$17,428 for suggestion #2. This means a reduction in TCO of 14.8% and 30.0% for suggestions #1 and #2, respectively. The performance per TCO metric leads to a gain of 6.7% for suggestion #1 and a loss of 10.0% for suggestion #2.

- **Performance per TCO^2 :** If total cost of ownership is more important than performance, performance per TCO-squared might be an appropriate metric. Using this metric, it is clear that there is a big benefit in using our two suggestions compared to the hardware vendor's suggestion, with a respective gain of 25.2% and 28.5%.

Note that total (both static and dynamic) power consumption is likely to be reduced even more because of reduced operating temperature which reduces leakage power consumption. In other words, the reduced cost factors mentioned above are pessimistic cost savings; actual savings in power consumption and total cost of ownership will be higher.

In summary, this case study illustrated evaluating hardware design choices in the data center, enabling service providers, data center owners, as well as system architects to make trade-offs taking into account end-user performance of a Web 2.0 workload.

4.7.2 Software optimizations

Whereas the first use case considered a hardware design trade-off, our second use case illustrates the potential for driving software trade-offs and analyses. The reason why this is valuable is that setting up such experiments in a live data center is considered to be too risky because it might interrupt normal operation. Our approach on the other hand allows for setting up such experiments in a controlled environment while being able to apply real-life user requests.

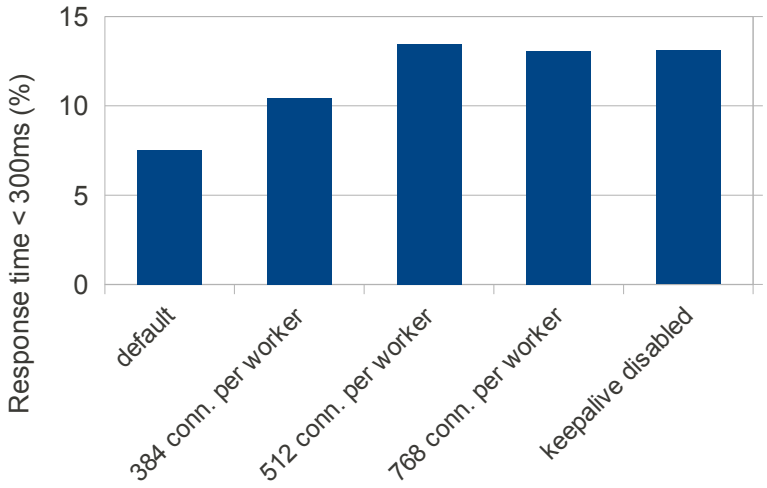

Figure 4.17: Increase in number of requests handled under 300 ms for different NGINX configurations, compared to the Apache Web server.

In this case study, we analyze the performance for an alternative Web server software package. As mentioned before, Netlog uses the Apache Web server software to process all user requests on the Web servers. Another well-known web server software package is called NGINX – High performance Web server with low memory footprint.[18] In Figure 4.17, we show the percentage increase in the number of requests that were handled under 300 ms, the chosen metric for this case study. We compare different NGINX configurations to the standard Netlog Apache configuration. On the horizontal axis we distinguish several NGINX configurations, starting with the default configuration on the left side.

We observe that replacing Apache by a default NGINX setup increases the number of requests handled under 300 ms by 7.5%. This number gets up to 13.5% when tuning the number of connections per worker thread. We also disabled the HTTP keepalive feature, but conclude that there is no performance difference in disabling this feature.[19]

NGINX reduces response times as perceived by the end user, thereby increasing customer satisfaction. This will lead to more users visiting the social network site, leading to an increase in company profit. System engineers and software developers can easily study other software tweaks or parameter tuning for maximizing performance in the data center by using the proposed method.

[18] http://www.nginx.net/

[19] The 'keepalive' feature is used for actively maintaining a connection between a client and a server.

4.8 Related Work

4.8.1 Data center workloads

A number of studies have been conducted recently to understand what hardware platform is best suited for a given data center workload. In all of these setups, a single server is considered — the study focuses on leaf-node performance — and/or microbenchmarks with specific behavior are employed. In contrast, this work considers a setup involving multiple physical servers running real workloads, and we focus on end-user performance.

Kozyrakis et al. [43] consider three Microsoft online services, Hotmail, Cosmos (framework for distributed storage and analytics) and the Bing search engine, and their goal is to understand how online services and technology trends affect design decisions in the data center. They collect performance data from production servers subject to real user input, and in addition, they set up a slightly modified version of the software in a lab setup in order to perform stress tests for evaluating individual server performance under peak load. Our work differs from the Kozyrakis et al. work in two important ways: (i) we consider a different workload (Web 2.0 social networking), and (ii) our methodology is very different: our lab setup includes multiple servers, running unmodified production software supplied with real-life user input; in addition, we focus on end-user perceived performance.

Lim et al. [47] consider four Internet-sector benchmarks, namely Web search (search a very large dataset within sub-seconds), webmail (interactive sessions of reading, composing and sending e-mails), YouTube (media servers servicing requests for video files), and mapreduce (series of map and reduce functions performed on key/value pairs in a distributed file system). These benchmarks are network-intensive (webmail), I/O-bound (YouTube) or exhibit mixed CPU and I/O activity (Web search and mapreduce). Lim et al. reach the conclusion that lower-end consumer platforms are more performance-cost efficient — leading to a 2× improvement relative to high-end servers. Low-end embedded servers have the potential to offer even more cost savings at the same performance.

Andersen et al. [1] propose the Fast Array of Wimpy Nodes (FAWN) data center architecture with low-power embedded servers coupled with flash memory for random read I/O-intensive workloads. Vasudevan et al. [75] evaluate under what workloads the FAWN architecture performs well while considering a broad set of microbenchmarks ranging from I/O-bound workloads to CPU- and memory-intensive benchmarks. They conclude that low-end nodes are more energy-efficient than high-end CPUs, except for problems that cannot be parallelized or whose working set cannot be split to fit in the cache or memory available to the smaller nodes —

wimpy cores are too low end for these workloads.

Reddi et al. [65] evaluate the Microsoft Bing Web search engine on Intel Xeon and Atom processors. They conclude that this Web search engine is more computationally demanding than traditional enterprise workloads such as file servers, mail servers, Web servers, etc. Hence, they conclude that embedded mobile-space processors are beneficial in terms of their power efficiency, however, these processors would benefit from better performance to achieve better service-level agreements and quality-of-service.

Meisner et al. [51] introduce PowerNap, an energy-conservation approach where the entire system transitions rapidly between a high-performance active state and a near-zeropower idle state in response to instantaneous load. However, the authors focus on a special type of server enclosures only, called blade servers.

Zhao et al. [81] study the impact of using HipHop, a static compiler for the dynamic PHP language, on some simple microbenchmarks and on the Facebook workload. The PHP language is a popular language that is often used in scale-out Web applications. As mentioned before, PHP is an interpreted language and it is known that PHP incurs a lot of overhead. The authors show that the use of HipHop is up to 5.5x more efficient than traditional PHP execution. This technique might be beneficial for the Netlog workload as well.

Lotfi-Kamran et al. [48] consider emerging applications in scale-out data centers. They show that performance of scale-out workloads that operate on large datasets is maximized through a modestly-sized last-level cache that captures the instruction footprint at the lowest possible access latency instead of microarchitectures that waste silicon on excessively large caches. We must note that the results of Lotfi-Kamran et al. are very much dependent upon the workload being considered — results might be very different for the Netlog workload.

Hoffmann et al. [30] present a system for dynamically adapting application behavior depending on load and power fluctuations. The authors show that their benchmark applications execute responsively in the face of power caps that would otherwise significantly reduce performance. They also prove that dynamically adapting application behavior can help dealing with reducing the number of machines needed to handle load spikes, reducing power usage and TCO.

4.8.2 Sampling

Sampling is not a novel method in performance analysis. Some of the prior work mentioned above focuses on leaf-node performance, an example of

sampling in space. Sampling in time is heavily used in architectural simulation. Current benchmarks execute hundreds of billions, if not trillions, of instructions, and detailed cycle-accurate simulation is too slow to efficiently simulate these workloads in a reasonable amount of time. This problem is further exacerbated given the surge of multi-core processor architectures, i.e., multiple cores and their interactions need to simulated, which is challenging given that most cycle-accurate simulators are single-threaded.

Sampled simulation takes a number of samples from the dynamic instruction stream and only simulates these samples in great detail. Conte et al. [16] were the first to use sampling for processor simulation. They select samples randomly and use statistics theory to build confidence bounds. Further, they quantify what fraction of the sampling error comes from the sampling itself (sampling bias) versus the fraction of the error due to imperfect state at the beginning of each sample (non-sampling bias or cold-start problem). Wunderlich et al. [79] employ periodic sampling and very small samples while keeping the cache and predictor structures 'warm', i.e., cache and predictor state is simulated, while fast-forwarding between samples. Follow-up work by Wenisch et al. [77] introduces the concept of checkpoint-based sampling. Each checkpoint or so-called 'Live point' is a storage-efficient implementation that saves the current state of the processor, including caches, predictors, etc. Checkpoint-based sampling allows individual performance measurements to be simulated independently, reducing simulation time drastically.

Whereas both the Conte et al. as well as the Wunderlich et al. approaches select a large number of samples and rely on statistics to evaluate the representativeness of the samples, Sherwood et al. [71] employ knowledge about program structure and its execution to determine representative samples. They collect program statistics, e.g., basic block vectors (BBVs), during a profiling run, and they then rely on clustering to determine a set of representative samples. The approach taken in this work is similar to Sherwood et al. although we take different workload statistics as input to the sample selection algorithm, while considering server workloads rather than CPU workloads.

4.9 Conclusion

In this chapter, we presented a case study in which we characterized a real-life Web 2.0 workload and evaluated hardware and software design choices in the data center. Our methodology samples the Web 2.0 workload both in space and in time to obtain a reduced workload that can be replayed, driven by input data captured from a real data center. The reduced work-

load captures the important services (and their interactions) and allows for evaluating how hardware and software choices affect end-user experience (response times).

The real-life Web 2.0 workload used in this work is Netlog, a popular and commercially deployed social networking site with a large user base in Europe. We explored hardware trade-offs in terms of core count, clock frequency, HDD versus SSD, etc., for the Web, memcached and database servers, and we obtain several interesting insights, such as the Web servers scale well with core count, and end-user response times are inversely proportional to Web server CPU frequency; an SSD reduces the longest response times by around 30% over an HDD in the database servers, which may or may not be justifiable given the significantly higher cost for SSD versus; memcached servers show low levels of CPU utilization, and are both memory and network-bound, hence, hardware choice should be driven by the cost of integrating more main memory in the server. Further, we presented two case studies illustrating how the method can be used for guiding hardware purchasing decisions as well as software optimizations.

Chapter 5

Database Cloning

The database management system is an important component of a contemporary Web 2.0 workload, yet improving its performance is challenging. Evaluating hardware and software alternatives and trade-offs in a production environment is complicated and might not always be possible; copying (part of) a database to an offline environment might not be feasible either, particularly because of intellectual property and privacy issues. In this work, we propose a framework for generating synthetic but representative database clone workloads.

5.1 Introduction

Optimizing Web 2.0 performance is non-trivial because of the complex interplay between the many software components (load balancers, web, memcached and database servers), the scale of the data center, non-determinism, bursty user requests, etc. An important software component of a Web 2.0 workload is the database management system. Requests that do not happen to be cached by the memcached servers need to get the data from the database servers. Hence, the requests that need to go to the database server are arguably the slowest — even though the database may reside in memory, as is typically the case for contemporary Web 2.0 workloads in order to deliver good interactive performance to the end user. Fine-tuning the database management system and the hardware it runs on is challenging in itself as it is a multifaceted optimization endeavor including criteria related to cost, performance, scalability, dependability, availability, power consumption, etc. Contemporary database systems have complex configuration possibilities, and identifying the optimal software configuration is difficult. Further, the best configuration might even change over time as new features are being offered by the services that they support. Making these trade-offs is non-trivial because of the complexity of the system, and making these trade-offs in an online environment makes

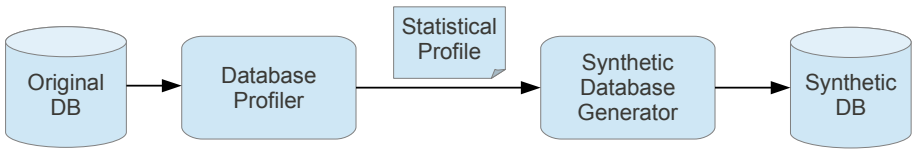

Figure 5.1: Overview of the synthetic database generation framework.

it even more challenging because it may affect user experience. Further, performing offline evaluations on a copy of the database comes with its own issues. Copying a large database onto a test platform may be time-consuming and costly, if at all possible, because of the size of the database. In addition, it may not be desirable to do so because of privacy issues. For example, sharing (part of) the database with third parties (e.g., a database performance consultancy firm, or a processor manufacturer or system integrator interested in optimizing hardware for their client's workloads) is likely to be impossible without anonymizing the database. However, anonymizing the database to make sure no sensitive (business and/or personal user) information is leaked is hard to automate and is likely to be error-prone.

In this work, we present a framework for creating a synthetic duplicate or clone of an existing database, see Figure 5.1 for a high-level overview. In the first step, a database profiler constructs a statistical profile that describes the original database in a statistical manner through a collection of distributions. The statistical profile serves as input to the synthetic database generator. This generator then builds a synthetic database that can be used for various experimental purposes. Synthetically generated databases have a number of key benefits.

- **Ease of duplication:** The profile can be collected with limited overhead. Once the profile is collected, a synthetic duplicate can be reconstructed offline. There is no need to copy hundreds or thousands of gigabytes of data from the production environment to a test environment.

- **No sensitive information:** The statistical profile captures high-level information about the data in a statistical manner. No actual user information or company related data is stored in the profile. This enables sharing the synthetic database with third parties performing representative performance analyses without having to worry about IP or privacy issues. For example, a database consultancy firm could run analyses and give feedback on how the database management system (DBMS) should be tuned. Likewise, processor manufacturers and system integrators can run performance analyses using a synthetic database clone.

- **Repeatability:** Doing performance measurements and experiments in a live environment is complicated as the state of the database continuously changes. Hence, it is difficult to reproduce experimental results. Further, performance comparisons between design alternatives can only be done in a meaningful way through sophisticated statistical analysis. Synthetic databases provide a solution for this problem as they allow for running multiple experiments starting from the same initial conditions in an offline environment.

- **Scalability:** By adjusting the statistical profile, it is very easy to generate synthetic databases with specific characteristics of interest, including very large databases, complex inter-table relationships, novel data types, etc. This enables answering interesting what-if questions. In particular, it can be used to do database scalability analysis, i.e., a Web 2.0 company may want to study the effect of an increasing number of users, leading to a growth in database size and how this affects user response times and server load.

We illustrate our framework on a database taken from the same real-life Web 2.0 workload as in the previous chapter, Netlog. We demonstrate that we can generate a synthetic, anonymized clone from the real database in limited time, i.e., generating a statistical profile from a 3 GB database shard takes around 10 minutes, and generating a synthetic, anonymized clone takes around one hour; further, anonymizing and matching up a recorded query log with the synthetic clone is straightforward once we have the anonymized database. The synthetic clone generates similar performance behavior as the original database: we find that queries sent to the synthetic clone are accurate within 0.95% on average compared to the original database, and the 90% percentile of the response time distribution is estimated with an error of 1.38%. We further illustrate the usage of synthetic database clones for both hardware and software performance studies. We show that the synthetic clone leads to the same conclusions when evaluating hardware design trade-offs when changing CPU clock frequency, enabling/disabling hardware prefetching, evaluating hard-disk drives versus solid-state disks, and comparing processor families. We evaluate several software trade-offs using the synthetic clone while evaluating database scalability, exploring query cache size, and comparing storage engines (InnoDB versus MyISAM).

We envision several use cases for this framework. Database performance analysts, administrators and engineers can use the framework to evaluate hardware and software alternatives and trade-offs in an offline test environment. External consultancy firms can even run off-site performance analyses as the database is anonymized. Our own primary interest is to eventually use this framework for generating representative but

anonymized workloads for architecture and code optimization research. This work is just a first step towards this end. We now have the ability of anonymizing a real-life database workload, yet the workload is still long running and thus not amenable to simulation using slow architectural simulators. The next step is to reduce the size of the workload, either by reducing the size of the database or by selecting a number of representative queries or both, so that the workload can be simulated using detailed processor simulators in a reasonable amount of time.

The remainder of this chapter is organized as follows. Section 5.2 introduces the basic concepts of (relational) database management systems. We present the statistical profile that is used to summarize the database in statistical terms in Section 5.3. Section 5.4 describes how the synthetic database is generated and in Section 5.5 we describe how the synthetic query log is generated. In Section 5.6 we validate the presented techniques. We then provide several use cases that illustrate the usefulness of synthetically generated databases for hardware and software studies in Sections 5.7 and 5.8, respectively. Section 5.9 illustrates the pitfall when using simple databases such as OSDB for evaluating DBMS performance, which reinforces the need for a rigorous framework like the one presented in this Chapter. Finally, we describe related work in Section 5.10, and we conclude in Section 5.11.

5.2 Relational Databases

Before describing how we characterize a database in a statistical manner and how we generate a synthetic database clone from this profile, it is important to understand the fundamentals of relational databases, which is the purpose of this section.

The relational data model is the predominant choice for storing data in a database. A relational database is organized as a set of tables, in which each row represents a data tuple and each column represents an attribute; a table basically stores all attributes for all data tuples and represents a so-called 'relation'. A relational database is the most widely-used database type, for a number of reasons.

- **Wide adoption:** A number of big vendors like Oracle, IBM and Microsoft offer enterprise database systems and there is a large community supporting both commercial (Oracle, DB2, SQL Server) and open-source database systems (MySQL, PostGreSQL, SQLite).
- **Intuitive data model:** The relational model structures data in an intuitive manner, thereby avoiding complexity. To minimize redundancy and dependency, tables are typically normalized meaning that redun-

dant data is eliminated by dividing large tables into small tables and by defining relationships between the tables. This eases the process of inserting, deleting and modifying fields to just one table and then propagate those changes to the rest of the database through the defined relationships.

- **Data retrieval:** All relational database management systems (RDBMS) support an easy-to-use human-readable language for querying the database, namely SQL.
- **Flexibility:** A relational database is extensible by nature, providing a flexible structure for changing requirements and adding additional data. The relational model permits easy changes to the database structure without impacting the rest of the database, i.e., adding or removing tables does not compromise the rest of the database.
- **Security:** A relational database supports fine-grained access permissions and supports the concepts of users and user rights.

5.2.1 Column attributes

The database of a Relational Database Management System (RDBMS) consists of multiple tables, with each table having one or more columns. The columns hold the actual data, and each column has a specific data type. (We describe common data types in Section 5.2.2.) Further, there are a few special column attributes worth mentioning at this point as they form the foundation of a relational database.

- ***PRIMARY KEY*:** This column attribute means that the respective column uniquely identifies the rows in a table. Primary keys are guaranteed to be unique, and are user-defined or autogenerated by the DBMS.
- ***FOREIGN KEY*:** A foreign key is used to express relationships between tables. A foreign key will match a primary key of another table meaning that rows in the two tables with the same foreign and primary key, respectively, are related.

  Our test setup uses *MyISAM* as a storage engine.[1] *MyISAM*, however, does not have support for defining foreign keys explicitly; instead, foreign keys are defined implicitly. In Section 5.3.3, we will show how to detect relationships between tables with implicitly defined foreign keys.

[1] http://dev.mysql.com/doc/refman/5.0/en/myisam-storage-engine.html

- ***INDEX*:** To increase query performance, a special data structure, called an index, can be used. The performance impact of using indices is significant [46]. It is therefore important that our synthetic duplicate uses indices the same way as the original database; one of our case study illustrates this.

- ***UNIQUE*:** The unique attribute is used to express that only unique values can be used in a column. A more complex use of this constraint allows multiple columns to be combined into a single unique constraint. For example, the column *ADDRESS* and *NAME* of a user table could be specified to have a collective, unique constraint, i.e., although multiple people may live at the same address and although multiple people may have the same name, multiple persons having the same name *and* living at the same address is (almost) impossible.

5.2.2 Data types

In order to build a statistical profile of the original database, it is important to understand what data types are supported by the RDBMS. The following is a brief summary of some of the most common data types available in most database management systems, including MySQL which we use in our setup.

- ***CHAR, VARCHAR*:** These data types represent a number of characters. The number of bytes needed by a *CHAR* is fixed, whereas the number of bytes for a *VARCHAR* data element depends on the value that is being stored.

- ***TEXT*:** This data type is used to store large amounts of text. In most database systems, text objects are stored separately from the table, not as part of a row, which implies that retrieving data from a column with the *TEXT* data type will incur more overhead because of an additional indirection.

- ***ENUM, SET*:** *ENUM* is a string data type with a single value chosen from a list of permitted values (e.g., small, medium, large). The list of permitted values is specified during column creation. If a value is inserted that is not included in the possible values, an empty string is inserted.

  The *SET* data type is a similar string data type that can have zero or more values. As with *ENUM*, each of these values must be chosen from a list of permitted values.

- ***INT, FLOAT*:** These data types are used to store integer or floating-point values. There are different types of *INTEGER* values: *TINYINT*,

SMALLINT, *MEDIUMINT*, *INT*, and *BIGINT*, representing 1, 2, 3, 4 and 8 bytes of storage, respectively.

There are two floating-point types: *FLOAT* and *DOUBLE*, using 4 and 8 bytes, respectively. The above data types are used in the MySQL database system. Other database management systems might not support all of these or might require different storage demands.

- ***DATE, TIMESTAMP*:** The *DATE* data type is used to store a date; the smallest granularity is one day. If a finer granularity is needed, one can use the *TIMESTAMP* data type which offers one-second precision.

5.3 Statistical Database Profile

Building a statistical profile of the original database is the first step in our synthetic database generation framework, as previously shown in Figure 5.1. We want this profile to be small but as accurate as possible — it should accurately summarize the data in the database in a statistical manner — without storing sensitive information about the company, its business and its users.

5.3.1 Database scheme

There is some information available in the original database scheme as it describes the structure of all the tables in the database, including column names and column data types. Relationships between tables in the database scheme may be revealed as well, for example through foreign keys — not the case in our environment though as it uses MyISAM, as mentioned before. Finally, it may provide information about column constraints, keys and indices. The database scheme may thus reveal business logic. In order to obfuscate as much business logic as possible, we replace table and column names by anonymized names by generating random strings of the same length. Yet, the statistical profile keeps track of the number of tables in the database, as well as the number of columns and their data types; this is important in order to achieve similar performance characteristics for the synthetic clone compared to the original database.

5.3.2 Statistical profile of column data

For each table in the database, we also collect a statistical profile regarding its columns. We describe a column's data properties in a statistical manner using distributions. Different types of profiles are kept per data type.

- ***CHAR, VARCHAR*:** Since *CHAR* values are of fixed size and we already know the size from the database scheme, we only store the number of unique values appearing in the column. As mentioned before, *VARCHAR* data types have variable length; the data is more likely to be unique and of different size. To mimic the same behavior with the synthetic clone as the original database, we build a histogram to capture the different lengths in the column. We use 2048 bins to represent its distribution. Given that the *VARCHAR* data type has a maximum length of 64 K characters, the first bin thus counts the number of column entries with 0 to 32 characters, the next bin counts the number of column entries with 32 to 64 characters, etc.

- ***TEXT*:** This data type is similar to the *VARCHAR* type. We build a histogram that counts the number of occurrences of each length — this is done the same way as for the *VARCHAR* data type.

- ***ENUM, SET*:** The *ENUM* and *SET* data types take values from a predefined set of values. We anonymize these values in the statistical profile. We also capture the distribution amongst the value set and the number of empty values, i.e., we have a count saying how frequently each value is used and how frequently no value was chosen.

- ***INT, FLOAT*:** We store the minimum and maximum value of these data types across the column. If a column with the *INT* data type is used as a foreign key, we also store a histogram of frequency of occurrence for each data value. The histogram consists of 20 K buckets divided evenly between the minimum and maximum values. The reason for capturing a histogram for *INT* data types is that this data type is used to define relationships between tables through primary and foreign keys. Preserving the number and nature of inter-table relationships has a significant impact on database performance as perceived through response times.

- ***DATE, TIMESTAMP*:** In the statistical profile, we store the earliest and latest date. We also store the number of unique dates and the percentage of nil values (0000-00-00) appearing in the column.

5.3.3 Statistical profile of table relationships

As mentioned before, foreign keys play an important role in a relational database because they represent relationships between tables. We distinguish three ways for detecting relationships between tables.

Database scheme

Most database management systems implement table relationships using foreign keys. Explicit foreign keys allow one to easily determine relationships from the database scheme. However, as mentioned in Section 5.2.1, we could not extract foreign key information from our test database because it uses MyISAM as its storage engine, which does not make foreign keys explicit. Relationships in our case are implicit and appear through the same column data values with a primary key from another table.

Analyzing query log

An alternative approach is to extract table relationship information from analyzing the queries that are executed on the database. Suppose the following query is performed on the database:

```
SELECT * FROM PICTURES INNER JOIN USERS
ON USERS.userId = PICTURES.userId
WHERE USERS.name = 'Bob'
```

As we can see from this example, table *USERS* is joined with table *PICTURES* through the *userId* column. This way, we can conclude that there is a relationship between the two tables, namely through the *userId* column. Unfortunately, this technique for detecting relationships does not work on our test database either — and may not work in general — because the queries in our database workload do not use *JOIN* statements between tables across database shards.[2]

Data analysis and clustering

The two previous techniques do not work on our database, for the reasons mentioned above. Hence, we need an alternative heuristic approach for detecting relationships between tables. Primary keys are known from the database scheme, i.e., there is one primary key per table. The question now is to find foreign keys. We do this through a clustering algorithm. The purpose of the clustering algorithm is to group columns from different tables for which there is a high probability that they are related to each other. These clusters then presumably contain the foreign keys that relate to the primary keys in the cluster. It is possible that one cluster contains

[2]Database sharding is a horizontal partitioning technique in which every x rows of a table are stored in a separate database. For example, all information about users 1 to 100 is stored on database server 1, information about users 101 to 200 is stored on database server 2, etc. Each shard is typically stored on a different physical server. This technique makes it easy to scale out to a large number of users.

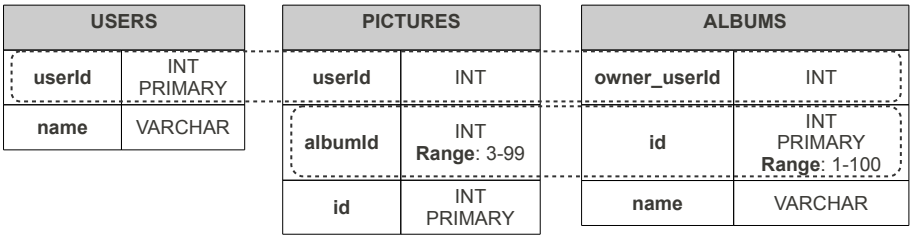

Figure 5.2: Clustering example: Based on the name and data type, we can detect that the columns *userId* and *owner_userId* are related to each other. Column *albumId* of table *PICTURES* and column *id* of table *ALBUMS* are of the same type and their data range is similar; hence we conclude that these columns are also related.

more than two columns from a given table, in which case we will have to decide which column, apart from the column holding a primary key, is the foreign key.

The clustering algorithm considers all possible combinations of columns and withholds column combinations if they fulfill to all three below criteria.

- **Name clustering:** Columns that refer to each other often carry the same or a similar name. For example, in Figure 5.3, column 'userId' from table *PICTURES* refers to column 'userId' from table *USERS*. A column combination is withheld if there is a match in column name in the original database (i.e., if a column name is a substring of the other column name).
- **Data type clustering:** If two columns have different data types, it is very unlikely that they will refer to each other. Hence, we only cluster columns with the same data type.
- **Value range clustering:** We try to find related columns by checking their value ranges. We compare minimum and maximum values of a pair of columns and we decide that the likelihood of being related is high if these values correspond well. Two columns are said to be within similar if the relative difference in range is smaller than a given threshold (3% in our case).

Figure 5.2 shows an example to illustrate how column clustering works. We can see that the first columns of each table are related — they all have 'userId' in their names. The second column of the *PICTURES* and *ALBUMS* table are related as well; we detect this because their value range is very close to each other.

The above algorithm for detecting inter-table relationship is based on heuristics, hence it is imperfect — nevertheless, we found it to work well for our test database. In addition, our framework offers its users the ability to manually verify whether the proposed clustering makes sense, and which clusters should be accepted and which not. (Note that the user here refers to the person anonymizing the database by collecting a statistical profile and generating a synthetic database clone. The user of the synthetic clone only gets to see the anonymized data and is unable to reverse-engineer the anonymized data.)

The next step is to convert the clusters into appropriate primary and foreign keys — we will use the primary and foreign key column information during synthetic database generation, i.e., we will first populate the primary key columns with unique data values before populating the foreign key columns as they which refer to the primary key columns. One of the columns in each cluster will be chosen as the primary key, and the others will act as foreign key columns. We distinguish three different scenarios:

- If there is a single primary key column in the cluster, then this column is the primary key and the other columns are foreign keys.
- If more than one column is a primary key in its table, then we will randomly pick one to be the primary key column; the other columns are then foreign keys.
- If there are no primary key columns in the cluster, we will denote the column that has the most unique values as the primary column and the others as foreign key columns.

5.4 Synthetic Database Generation

Having collected the statistical profile, we can now build a synthetic duplicate of the original database.

5.4.1 Overview

Generating a synthetic database is done as follows. We first generate all the tables. This includes establishing the number of rows, columns, column data types, (anonymized) table and column names, etc. We then define the primary keys in all of the tables, i.e., one column is chosen as the primary key per table. Once we have defined the primary keys, we then determine the foreign key columns which define the inter-table relationships. Finally, we populate the columns with anonymized data. We describe the synthetic database generation process in more detail in the next few sections. We

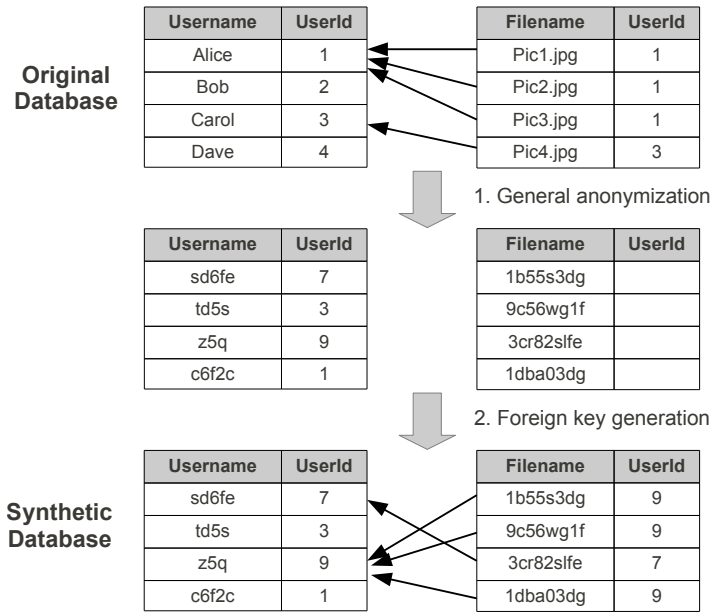

Figure 5.3: Example illustrating anonymization and foreign key generation. In this example we show two tables, *USERS* (left) and *PICTURES* (right). In the original database there is one user with three pictures, one user with one picture, and two users with no pictures. In the first step of the anonymization process, we anonymize the *Username*, *UserId* and *Filename* columns. We subsequently need to make sure that the foreign key distribution in the synthetic clone matches the histogram in the statistical profile. This is done by assigning three pictures to a single random user, one picture to another random user, and leave the other users without pictures.

describe how we generate primary keys, foreign keys and data values. Finally, we detail on how we satisfy unique constraints and how we deal with fragmentation.

Figure 5.3 shows an example illustrating how two tables of an original database are converted into synthetic versions in the clone database. Our technique makes sure that the foreign key distribution is the same in the synthetic duplicate as in the original database, and hence, the inter-table relationships are maintained in the synthetic database compared to the original database in a statistical sense.

5.4.2 Primary key generation

Primary key generation is done in a fairly straightforward way. As mentioned before, primary keys are defined using the *INT* data type. We know how many unique values there need to be, namely the number of rows in

the table, and we also know the range of data values in the column from the statistical profile. We then generate as many unique values as needed from the specified range, and randomly assign these unique values to rows in the table.

5.4.3 Foreign key generation

The generation of foreign keys needs two requirements to be fulfilled. First, the values in a foreign key column need to match with values of a primary key column in another table. Second, the distribution over these values needs to match the histogram stored in the statistical profile.

The following algorithm fulfills both requirements. We randomly choose a value from the primary key column the foreign key column corresponds to. (We verify that the value was not picked before.) We then determine how often this value should occur in the foreign key column; this is done using the histogram stored in the statistical profile. We generate a random number between 0 and 1, and use the inverse cumulative distribution to determine the number of occurrences of the value in the foreign key column. We then fill in this value as many times as just determined at random rows in the foreign key column. This process is iterated until all rows in the foreign key column are filled in.

5.4.4 Anonymized data value generation

Populating the tables with anonymized data is done as follows.

- ***CHAR, VARCHAR, TEXT*:** For all string data types, we insert a random alphanumeric string. The length of the string is determined by the histogram for *VARCHAR* and *TEXT* columns; the length of a *CHAR* column is determined by the length stored in the statistical profile. We also take the number of unique values into account and generate the same number of unique values as in the original database.

- ***ENUM, SET*:** One of the permitted values will be picked and inserted in the synthetic database; the permitted data values are anonymized by generating random values. The probability of each value is determined by the respective distributions in the statistical profile.

- ***DATE, TIMESTAMP*:** We insert the same number of nil values as stored in the statistical profile. For the other values, we insert a random date between the minimum and maximum date.

	ALBUMS		
	Column	Range	
PRIMARY KEY	userId	10.000	Root column
	albumId	500	Non-root column
	gender	2	Non-root column

Figure 5.4: In this example, a primary key containing multiple columns is defined. This makes it hard to determine values, because the combination of all three fields has to be unique. There are 10,000,000 possible unique combinations and if the number of required unique values is close to 10,000,000, it will be hard for a naive implementation to generate the final few values.

- ***INT, FLOAT*:** A random value between the saved minimum and maximum is inserted. If this column is a foreign key, insertion is more complicated, as described in the previous section.

5.4.5 Satisfying unique constraints

As explained in Section 5.2.1, a table may have a *PRIMARY* constraint and multiple *UNIQUE* constraints. We need to make sure we do not violate these constraints during data value generation. A naive implementation of randomly picking data values might not work effectively if the number of required values is close to the number of available values. Figure 5.4 illustrates this in which three columns (userId, albumId, and gender) are joined to form one *PRIMARY* key. In this example, the maximum number of possible values equals 10,000,000, which is the cross-product of 10,000 userIds, 500 albumIds and 2 gender type; assume we need to generate close to 10,000,000 unique values, say 9,999,999. A naive implementation that randomly picks a cross-product of these three values might have a hard time generating unique values as more and more cross-product values have been generated. In particular, the algorithm might need to try many times to find another unique value towards the end. We therefore take a different approach, and essentially generate all possible cross-product values, from which we then randomly pick.

5.4.6 Fragmentation

In order to be able to have repeatable performance measurements with synthetically generated databases, we pay attention to fragmentation and how this may affect database performance. We distinguish two types of fragmentation.

- **File fragmentation:** If a database is running in a production environment for months or years, the tables might get fragmented because of the many *DELETE*, *INSERT* and *UPDATE* statements. MyISAM uses several techniques to reduce fragmentation. In MySQL, one can remove fragmentation by executing the *OPTIMIZE* command. We compared the performance difference before and after running the *OPTIMIZE* command and we did not observe a statistical difference. Hence, we conclude that fragmentation is low and that there is little impact on performance, at least for our particular database.

- **Disk fragmentation:** Another type of fragmentation might happen at the disk level: a file that is stored on disk might get fragmented over time as well. Using the *filefrag* tool, we observed that large database files were often fragmented into smaller pieces.[3] To get similar fragmentation effects on both the original and synthetic databases in our experiments, we create a new file system before running each test. As a consequence, by copying the original or synthetic databases to this new file system, the fragmentation is very similar, and hence fragmentation has little impact on overall performance. Moreover, it enables repeatable experiments.

5.5 Generating Synthetic Query Log

Using the synthetically generated database for performance analyses also requires that we have a synthetically generated query log to exercise the database. One could use a random query log, however, this is unlikely to be representative for a live production environment. Alternatively, one could use a synthetically generated query log that stresses a particular component of the database — a so-called stress test. For example, one could stress the system by having many users inserting or downloading particular items in the database. This could lead to some interesting observation with respect to the scalability of the database in terms of the number of users and bursty behavior.

In this work, we capture a real query log and derive a synthetic version from it, in order to be as representative as possible compared to the real query log. We use this approach for validating our framework in this work. We obtain timing information for the real query log by replaying it against the original database; we apply the same procedure for the synthetic query log on the synthetic database, after which we then compare the timings between the synthetic runs versus the original database runs.

The query log contains an entry for each query with all of its parame-

[3] http://linux.die.net/man/8/filefrag

ters. Obviously, the original query log does not match with the synthetic database clone, i.e., the queries are no longer meaningful because of the anonymization process during synthetic database generation. Hence, the query trace log needs to be adapted accordingly to reflect the synthetic database, i.e., we also anonymize the query log and we make sure it matches the synthetic database. This is done as follows. Every query in the original query log is inspected, and values that are used in the query are replaced by existing values from the synthetic database. Because the database is anonymized, the query log will now also consist of anonymized data values, and yet, the query log will be meaningful with respect to the synthetic database, i.e., the synthetic database will reply anonymized data in response to the queries. We maintain the structures of the queries during this process while anonymizing data values. For example, if an original query on the original database asks for all the pictures from a particular user, the anonymized query will ask for all pictures from a random user in the database. The result returned from the database will be a set of file names (the pictures); these file names are just random strings as they were anonymized during the generation of the synthetic database clone.

In case of an *INSERT* query, we save the value that is inserted in the database. We do not insert the original value, because this might conflict with other values in the synthetic database and it might defeat our goal of anonymizing the database. Instead we insert a randomly generated data value. Later on, if the value is being searched for by a later query in the query log, we will modify the later query and use the (random) value we have just inserted.

5.6 Validation

Having described how we generate a synthetic database clone and how we generate a series of synthetic queries to exercise the database, we now validate our approach by comparing the performance of the synthetic database against the original database. This is done by replaying the original versus synthesized query trace log on the original versus synthetic database, respectively, and by comparing their performance numbers. If the query response times on the original database correspond with those of the synthetic one, we can conclude that our synthetic duplicate accurately mimics the behavior of the original database.

5.6.1 Experimental setup

Before describing our validation results, we first detail on our experimental setup.

The real-life Web 2.0 workload we used is taken from the same social network application as in the previous chapter, Netlog. The databases in the Netlog data center are organized per language. We focus on the Slovene database, which consists of more than 100 tables, including user, video, music, chatting, friends and blogging content, for a total of 90 GB of data. The Slovene part is representative for Netlog's entire workload because it exhibits the same partitioning of servers as the rest of Netlog's workload. Also, we observe similar degree of activity and access behavior (access to profiles, photos, videos, etc.) for the Slovene language as for the other languages. The Netlog workload uses MySQL 5.1 as its database management system.[4] As mentioned before, the database is horizontally partitioned, i.e., rows of database table are held separately on different physical servers — a technique called sharding. For the Netlog workload, this means that different users are distributed across different physical machines which enables holding the database in memory while scaling out to large numbers of users. Each shard in the Slovene part of the Netlog workload consists of a 3 GB database (there are 30 3 GB shards for the Slovene language); we consider one such 3 GB database shard in our study.

To mimic real user traffic we captured a query trace log on the original infrastructure. The query trace log contains 90,000 queries — the selection of the references from a single day of operation (March 13, 2011) that access the database shard that we selected. This trace file is replayed using our own query replayer, written in Java, which executes all queries sequentially with a 10 ms think time between consecutive queries.

We conducted our experiments on two hardware platforms. One platform has a 3.2 GHz Sandy Bridge quad-core Intel Xeon E3-1230 processor with 16 GB of memory. It has both a regular rotating Western Digital Caviar Green disk[5] as well as an Intel Postville solid-state disk[6]. Our second platform has a two 2 GHz dual-core AMD Opteron 2212 processors with 8 GB of memory. It has a regular rotating Western Digital Caviar Blue disk.[7]

Collecting the statistical profile and generating the synthetic database from it can be done in limited time. It takes 10 minutes to extract the database scheme and collect the statistical profile from our 3 GB Netlog database shard. In a real-life setting, this profiling can be done during quiet hours so that users experience as little overhead as possible. The benefit over making a copy of the original database through 'mysqldump' is that it does not require the database to be locked; an approximate statisti-

[4] http://www.mysql.com/

[5] Western Digital WDC WD20EADS-00R: 2 TB disk running at 5.400 rotations per minute with a 32 MB cache.

[6] Intel SSDSA2M080: 80 GB Multi level cell solid state disk with 32 MB cache.

[7] Western Digital WDC WD5000AAKS: 500 GB running at 7.200 rotations per minute with a 16 MB cache.

cal profile is good enough for our purpose. This also implies that profiling the database can even be spread over time. Further, the statistical profile is as small as 3 MB — a thousand-fold reduction in storage compared to the original database — hence, in contrast to copying the entire database, it does not add much pressure on the network and storage infrastructure in the data center. Generation of the synthetic database takes about one hour. This is acceptable, because it only has to be done once. If one wants to run multiple tests, it is possible to make a copy of the synthetic database.

5.6.2 Results

Because of the statistical nature of the proposed framework, i.e., we generate the synthetic database based on statistics using distribution and we anonymize the query log, we validate the database workload cloning in a statistical manner. This is done by comparing the response time distribution on the original database against the response time distribution on the database clone. In other words, we run the (anonymized) query log on the (anonymized) database and time the response for each query. We then compute the response time distribution. Figure 5.5 visualizes the response time distributions for both the original database workload and its synthetic version; Figure 5.6 shows the same information using the cumulative distributions. The distributions match well, and the synthetic database also illustrates the bimodal distribution; both the original and the synthetic database workloads show peaks around 400 and 600 microseconds. The second peak represents more complex queries that involve a lookup on a column that does not have an index. The mean response time for the synthetic workload is within 0.95% of the original workload. The 90% percentile is estimated with an error of 1.38%.

5.7 Hardware Tuning

Having gained confidence in the representativeness of synthetically generated databases and query logs, we now use this approach to explore hardware and software trade-offs. In this section we consider a number of hardware experiments; the next section then focuses on software experiments.

We run the following experiments on recent server hardware. The CPU in our test server is a Sandy Bridge quad-core Intel Xeon CPU E3-1230, which supports Dynamic Voltage and Frequency Scaling (DVFS) and advanced architectural settings (i.e., hardware prefetching) can be configured. The server has 16 GB of main memory and is running Ubuntu Linux 11.04. In the following tests, we execute 90 K queries on the original and the synthetic database. We repeat each test at least five times and we present aver-

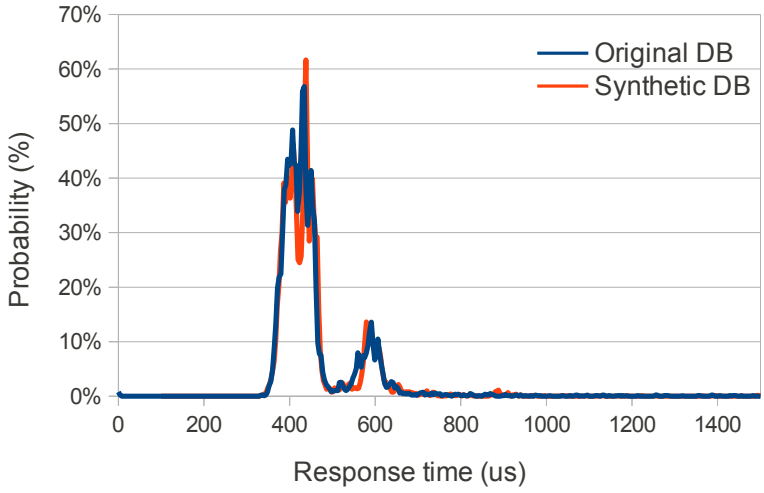

Figure 5.5: Validation of the database workload cloning. Comparing response time distributions (in microseconds) on the original versus cloned database.

age results along with confidence intervals with a 95% confidence level.

5.7.1 Clock frequency

The Intel Xeon CPU E3-1230 supports clock frequencies from 1.6 GHz to 3.2 GHz. In this section we show how CPU clock frequency affects query response time. As shown in Figure 5.7, increasing clock frequency from 1.6 GHz to 3.2 GHz reduces average query response time on the original database by 10%. Query response times get reduced by 8.5% on average on the synthetic database, which is close to what we observe on the original database. This result shows that the impact of varying clock frequency is rather low. This is to be understood intuitively as the database server is mostly memory-intensive. Since memory speed does not scale while scaling CPU clock frequency, the performance impact of CPU scaling is limited. Figure 5.8 shows a similar result for the 90% percentile of the query response time. Interestingly, increasing CPU clock frequency improves the slowest queries more than the average query (32% versus 10% improvement). The slowest queries involve more computation work which benefit from a higher clock frequency.

The important conclusion from this experiment is that a designer would reach the same conclusion using the synthetic versus the original database, namely changing CPU clock frequency affects query response time somewhat (around 10 percent) on average and improves the slowest responses more significantly (around 30 percent). This is potentially important information for Web 2.0 companies that seek at reducing their overall energy

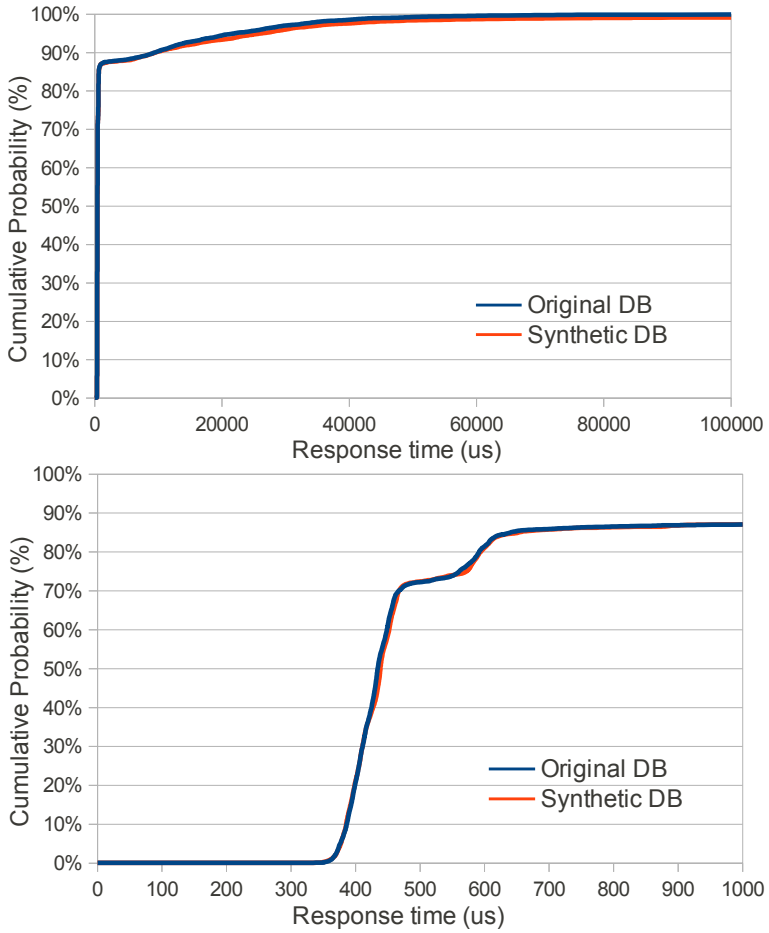

Figure 5.6: Validation of the database workload cloning. Comparing cumulative response time distributions (in microseconds) on the original versus cloned database. Entire cumulative distribution is shown on top; graph at the bottom is zoomed in.

consumption. If they decide to halve the CPU clock frequency to save more energy, then they can expect an average 10% (and up to 30%) drop in performance for the queries that need to go to the database server.

5.7.2 Hardware prefetcher

The Intel Xeon CPU E3-1230 (Sandy Bridge) has a hardware prefetcher that prefetches data into the last-level cache and possibly the L2 cache. The hardware prefetcher monitors read requests from the L1 (instruction and data) caches for ascending and descending sequences of addresses, and when a forward or backward memory reference stream is detected, the anticipated cache lines are prefetched [34]. It is possible to disable the hard-

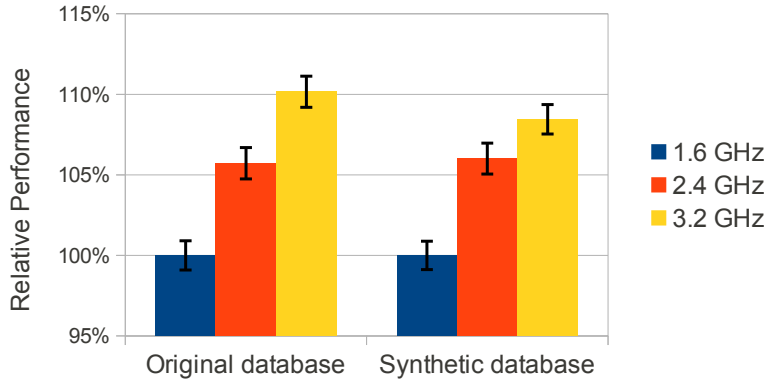

Figure 5.7: Average query performance as a function of CPU clock frequency for both the original and synthetic databases.

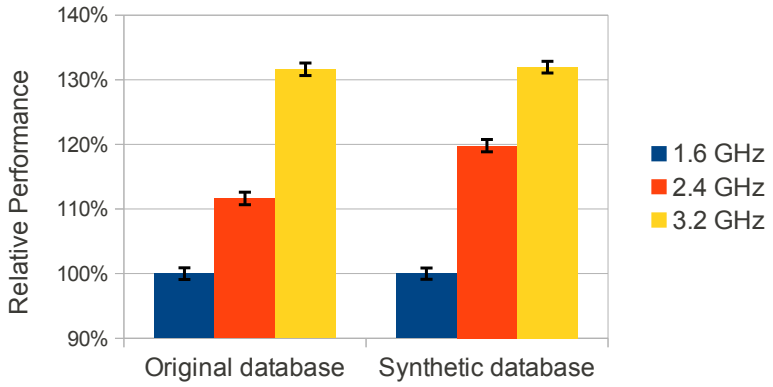

Figure 5.8: 90% percentile query performance as a function of CPU clock frequency for both the original and synthetic databases.

ware prefetch unit in our machine through the BIOS. By disabling and enabling this L2 hardware prefetcher we can measure the impact of hardware prefetching on response times. For both the original and synthetic databases, we find that enabling the hardware prefetcher improves query performance by a small percentage only, see Figure 5.9. Apparently, the memory access patterns are hard to predict by the hardware prefetcher, hence its impact on performance is limited. It is encouraging to observe that similar insights and results are obtained using the synthetic and original databases.

5.7.3 Solid-state disk

Solid-state disks are popular these days because of their high performance and low energy consumption. Their isolated performance has been stud-

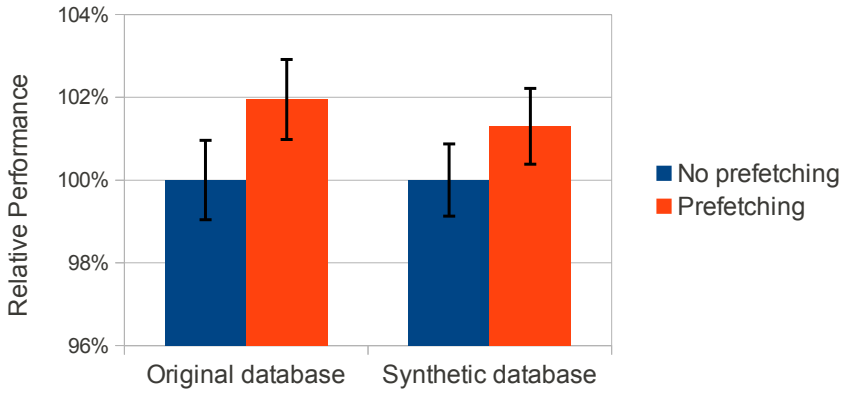

Figure 5.9: Impact on average query performance from hardware prefetching for both the original and synthetic databases.

ied before and the results look promising [17] [26]. However, solid state disks are still expensive these days, hence the question is whether solid-state disks yield a significant performance benefit for a Web 2.0 workload, and whether the higher cost is worth the performance benefit. To answer this question, we set up an experiment in which we replace the standard rotating Western Digital Caviar Green hard disk (HDD) by an Intel Postville solid-state disk (SSD).

For the purpose of evaluating this trade-off between HDD versus SSD, we limit the amount of physical memory in our machine to 1 GB. This will cause the 3 GB database shard to be stored partially in memory and partial on HDD/SSD. In other words, some requests will get the data from memory, others will get the data from HDD/SSD. This enables us to measure the impact of SSD versus HDD when at least a fraction of the requests cannot be serviced from memory.

As shown in Figure 5.10, the benefit of using a solid-state disk is large in case the database does not fit in physical memory. Query response times improve by 48% on the original database and by 44% on the synthetic database. Again, we conclude that the synthetic database accurately predicts the relative performance trends observed on the original database.

5.7.4 Comparing hardware platforms

In this section we compare our server platform (Intel Xeon E3-1230) with an (older) server that has two dual-core AMD Opteron 2212 processors. The AMD server has 8 GB of main memory and is running Ubuntu Linux 10.04. The results are shown in Figure 5.11. The newer Intel Xeon server platform, outperforms the AMD Opteron system by 18% according to the experiments with the original database; the experiments with the synthetic

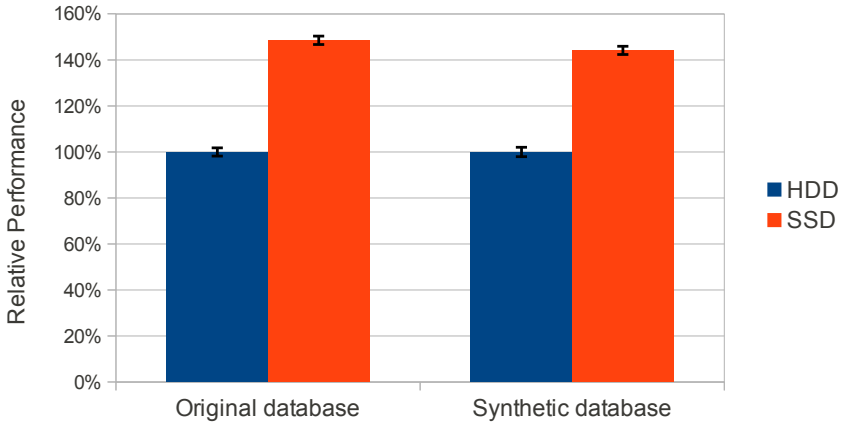

Figure 5.10: Impact on performance for a spinning hard drive versus a solid-state disk for both the synthetic and original databases.

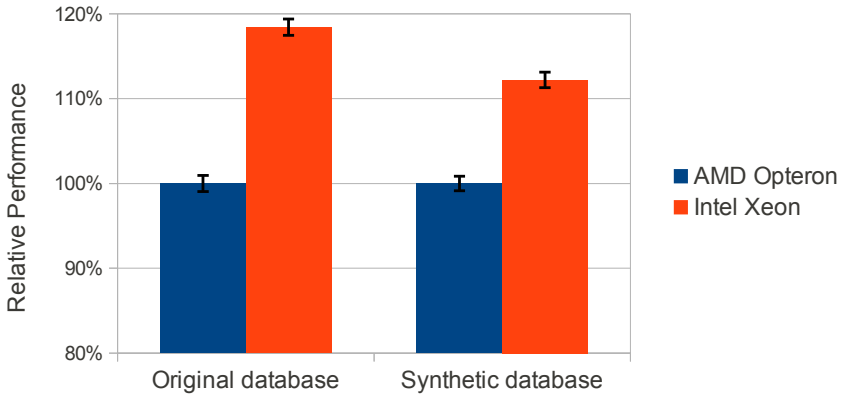

Figure 5.11: Comparing two hardware platforms: The Intel Xeon outperforms the AMD Opteron platform by more than 10% according to both the original and synthetic databases.

database report a performance improvement of 12%. This experiment illustrates how Web 2.0 companies and third parties can use synthetic databases to perform hardware performance evaluations.

5.8 Software Tuning

Configuring complex software, e.g., a database management system, is known to be a challenging task. In this section we show some case studies for tuning several software configurations using synthetic databases.

5.8.1 Database scaling

An interesting property of a framework for generating synthetic databases is that it allows for easily changing its characteristics and evaluating its impact on performance. For example, building a synthetic database that is larger than the original one can be done fairly easily by increasing the number of entries in each table, and adjusting the data values accordingly as well as the query log. This might be interesting for evaluating the scalability of the database model. In particular, given the popularity of social networking, it is important to understand how database upscaling affects performance in order to steer capacity planning.

In Figure 5.12 we show results for a synthetic database that is the same size as the original one, as well as results for a database that is twice the size of the original one. Doubling the size of the database was done by simply doubling the number of entries in each table; we populated the tables with random data following the distributions in the statistical profile. Separate synthetic queries were generated for both database sizes. As shown in Figure 5.12, doubling the database size increases the response times for a significant number of queries. On the other hand, there are also a lot of queries for which response time is not affected. A more detailed analysis revealed that queries that uses indices are largely unaffected by the database size, i.e., the indices allowed for quickly identifying the appropriate entries in the table. Queries that do not use indices are slowed down substantially when increasing the database size, the reason being that these queries require the table to be scanned sequentially. The insight is that in order to make the database scale better, it is important to use indices. Our tool enables database performance engineers to determine which types of queries do or do not scale with database size and for which types of queries the database should be optimized for.

5.8.2 Query cache

There are many complex configuration possibilities in modern database management systems. Hence, it is difficult and time-consuming to come up with an optimal configuration. Synthetic databases facilitate exploring various configuration options because they allow for running experiments starting from the same initial database state in an offline setting. Running similar experiments in an online production environments are challenging, if at all possible.

In this section we show one example in which we vary the query cache size of our DBMS. The query cache stores the content of *SELECT* queries. If an identical query is received later, the server retrieves the result from the query cache instead of parsing and executing the query again. When tables

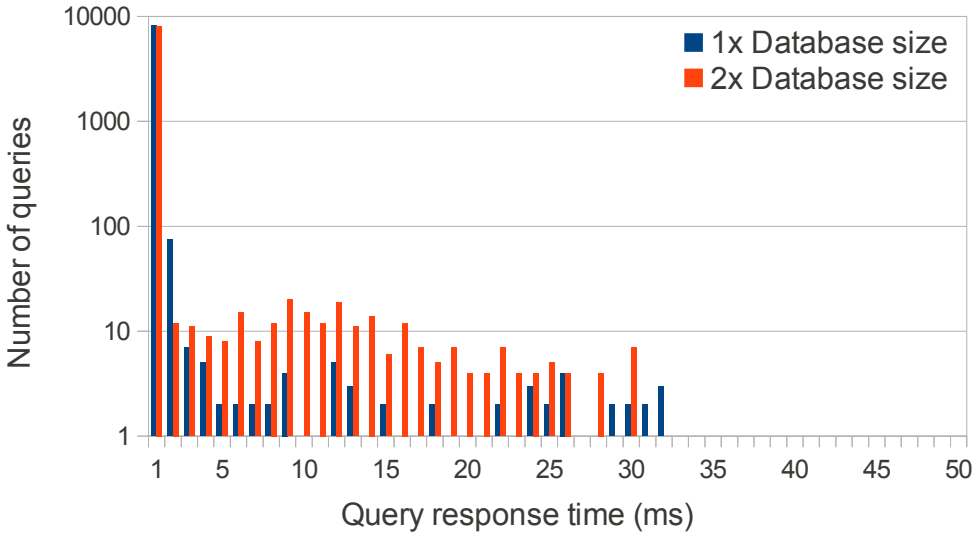

Figure 5.12: Scaling the database size by a factor of two introduces more queries with higher response times.

get modified, relevant entries in the query cache are flushed. The query cache is of interest to databases in which tables do not change very often and when the DBMS receives a lot of identical queries. The MySQL reference manual reports that the overhead of having a query cache active when there is nothing to be cached equals 13%, whereas single-row select queries are 238% faster if the query cache is active and used.[8] For our database, we find that a 8 MB query cache optimizes performance: it reduces average response time by 20% over no cache, see Figure 5.13. A database performance analyst may based on these experiments using synthetic databases suggest to set the query cache to 8 MB. We verified this using the original database and we found that an 8 MB query cache improves response time by 24% on average over no query cache — very close to the 20% predicted using the synthetic database workload.

5.8.3 Storage engine: InnoDB vs MyISAM

Our final case study explores the impact of the choice of storage engine on overall database performance. As mentioned throughout this chapter, the Netlog DBMS uses MyISAM as its storage engine. The question might come up whether it is worth replacing MyISAM with InnoDB.[9] InnoDB is a more recent engine that guarantees ACID properties.[10] Further, InnoDB

[8] http://dev.mysql.com/doc/refman/5.1/en/query-cache.html
[9] http://dev.mysql.com/doc/refman/5.0/en/innodb-storage-engine.html
[10] Atomicity, Consistency, Isolation, Durability

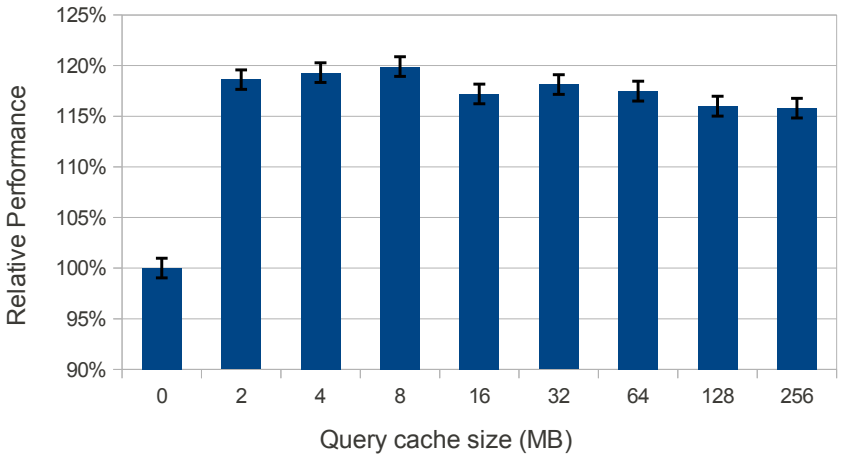

Figure 5.13: Impact on performance (query response time) for the MySQL built-in query cache as a function of its size.

uses row locking instead of table locking, which is faster for insert and update queries, but slower for select queries.[11] Finally, InnoDB has native support for foreign keys, transactions and rollback, which makes InnoDB a more complex storage engine compared to MyISAM. Answering this question in a live production environment is challenging. One option might be to route a fraction of the request stream from the production environment to a test environment. Statistically comparing the response times in the test environment against the production environment might yield the requested answer. Users whose requests are routed to the test environment will experience longer response times if the test environment yields poor performance. An alternative approach which does not affect end user experience is to use synthetically generated databases in an offline experimental environment.

In Figure 5.14 we show how query response time is affected by the two storage engines, InnoDB versus MyISAM. We find the difference to be small for queries that take less than 1 ms. However, MyISAM shows better performance numbers for queries that take longer than 1 ms. We note that most of these long-taking queries are complex select queries. As mentioned before, the overhead of table locking is lower than for row locking, which explains why the simpler MyISAM storage engine is performing better than the more complex InnoDB engine on average.

[11] http://dev.mysql.com/doc/refman/5.0/en/table-locking.html

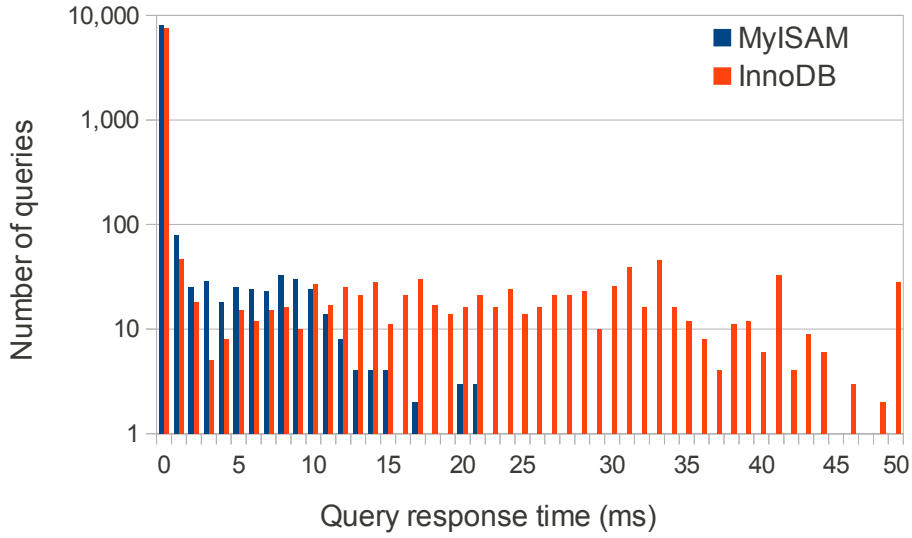

Figure 5.14: Comparing the performance impact of InnoDB versus MyISAM as a storage engine as a function of query response times. InnoDB introduces more overhead for insert operations, which leads to long-taking queries to take even longer.

5.9 Pitfalls in Using Simple Database Benchmarks

The Transaction Processing Performance Council (TPC)[12] provides a set of database benchmarks for evaluating hardware and software configurations for Online Transaction Processing (OLTP) and Decision Support Systems (DSS) workloads, which are considered the de facto standard today. Setting up these benchmarks is non-trivial though because it involves multiple machines (both clients and/or servers), disk arrays, and fine-tuning the DBMS. Further, it is extremely difficult to extrapolate performance numbers published by TPC to a particular database application and environment. This observation has inspired people to come up with simpler benchmarks, such as Open Source Database Benchmark (OSDB)[13] and SysBench[14], that are claimed to provide some insight in system performance while setting up a simple database (in case of OSDB) or without having to set up a database at all (in case of SysBench).

We make the case that synthetically generated benchmarks are in fact an interesting solution to the problem of evaluating hardware and software configurations for a given database. As mentioned throughout this Chapter, synthetic databases are easily set up, are representative for the

[12] http://www.tpc.org/
[13] http://osdb.sourceforge.net/
[14] http://sysbench.sourceforge.net/

real databases that they model, and enable repeatable experimentation in an offline environment. To illustrate the superiority of synthetic databases over simpler benchmarks, we now compare our synthetic database against OSDB, as an example of a simple database benchmarking tool. This is done through two experiments. In our first experiment, we scale CPU clock frequency, as done in Section 5.7.1, and measure its impact on query response times, see Figure 5.15. From the OSDB benchmark results, one would conclude that there is a significant performance benefit from higher CPU clock frequency. Synthetic databases show that the performance benefit is limited and smaller than what the OSDB workload suggests; this is in line with the experiments we did using the original database workload, see Section 5.7.1. In our second experiment, we run the OSDB benchmark on both of our server platforms, as done in Section 5.7.4. From the results in Figure 5.16, one would conclude that the Intel Xeon platform performs 36% better than the older AMD Opteron platform; our workload suggests that the performance benefit is more moderate as previously reported. These two experiments clearly illustrate that the OSDB database workload does not accurately capture the behavior of real-life database workloads of interest, and may lead to incorrect conclusions which may impact cost as well as end-user experience. In particular, OSDB uses a database consisting of 6 tables only, it does not model relationships between tables, and considers a limited number of basic queries only. A synthetically generated database using the proposed methodology on the other hand provides a more faithful representation compared to the real workload of interest. Our synthetic database consists of more than 100 tables, does model inter-table relationships, and runs anonymized queries based on real-life queries. This leads to a more complex, and more memory-intensive and less compute-intensive workload than OSDB, which explains why OSDB benefits more from increased CPU clock frequencies, both in Figures 5.15 and 5.16.

5.10 Related Work

We now describe related work in Web 2.0 and cloud workload performance analysis, and synthetic workload generation.

5.10.1 Web 2.0 and cloud performance analysis

Performance analysis for Web 2.0 and cloud workloads is challenging because of the scale of the system and the complex interactions taking place between the numerous software components and hardware servers. One approach is to construct a scaled-down version of a real workload. In the previous chapter, we characterized the real-life Netlog workload, the

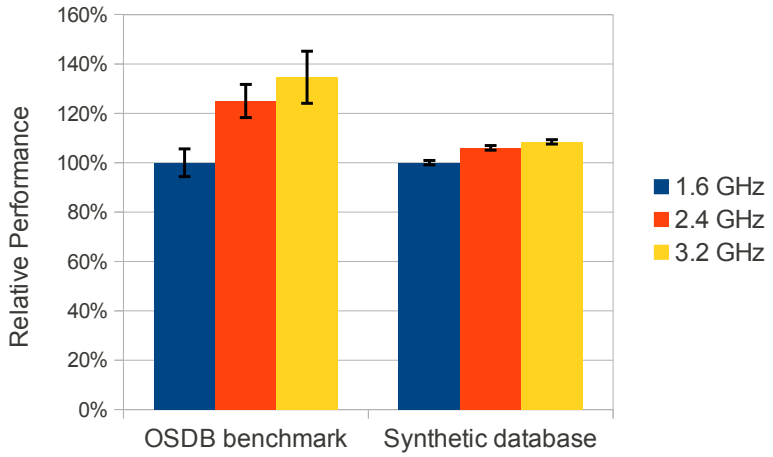

Figure 5.15: Evaluating how CPU clock frequency affects average response time for the OSDB and synthetic database workloads.

same workload used in this work. We therefore duplicated a subset of the workload — we selected one language — in an offline environment and we replayed real input traffic which we had captured over a period of four days. This setup enabled us to explore numerous hardware and software trade-offs and how they affect end-user experience as measured by response times. Setting up this environment was very time-consuming because we had to both make a copy of the software onto our test environment and we had to anonymize the database and input traffic. This was largely a manual process, which took us several man-months and which motivated the work presented in this work. By characterizing the database using a statistical profile and by generating a synthetic database and request stream, as we propose in this work, we no longer need to make a copy of the database and the anonymization is dealt with by construction. Having set up the infrastructure presented in this work, we are now able to characterize and duplicate a synthetic database clone and request stream in less than two hours, while yielding representative performance numbers.

Open Cirrus [3] is an open cloud-computing research testbed that was initiated by a collaborative group of researchers in both industry and academia. Open Cirrus' primary goal is to provide a distributed set of federated data centers as a testbed for system-level cloud computing research: it provides open-source software stacks and APIs, it enables systems-level research, it provides experimental data sets and it allows for studying application development for cloud computing.

An approach that is complementary to building a testbed or offline test environment, is to set up a simulation environment that allows one to perform more detailed performance analyses. Depending on the scope of the

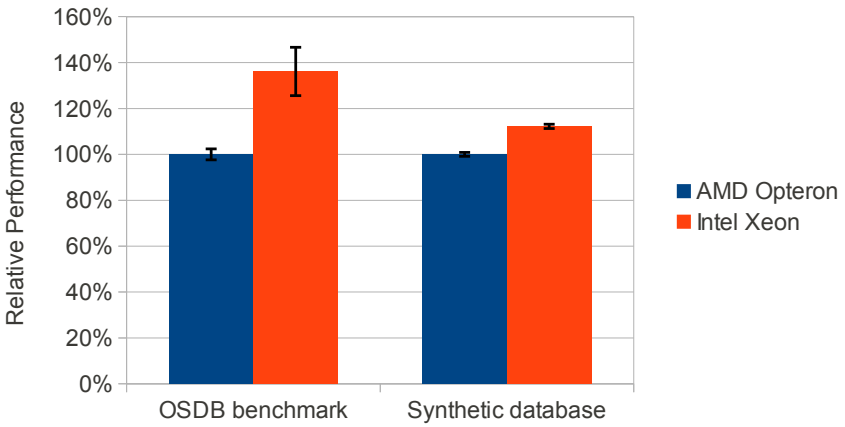

Figure 5.16: Evaluating how processor families affect average response time for the OSDB and synthetic database workloads.

study, one may opt for a detailed, but relatively slow, simulation strategy when focusing on an individual server or limited set of servers; alternatively, one may be interested in understanding system-level design tradeoffs over longer time scales and/or across multiple servers, and hence one may need a less detailed, and thus faster, simulation approach. CloudSuite [22] is a recently proposed benchmark suite for online service applications such as web search, social networking, and business analytics, and when run on a full-system simulator, such as Flexus [77] or gem5 [11], it enables performing detailed microarchitecture explorations for these emerging applications. System-level studies across multiple servers call for faster simulation strategies. COTSon [2] targets cluster-level systems consisting of multiple multi-core processor nodes connected through a network, i.e., it targets both scale-up (i.e., multi-core and many-core processor simulation) as well as scale-out (i.e., simulation of a multi-node cluster). COTSon uses the AMD SimNow full-system simulator to functionally simulate each node in the cluster. Each COTSon node further consists of timing models for the disks, network card interface and the CPU (i.e., processor and memory) [68]. VSim [69] uses time dilation through virtual machines to simulate multi-server setups at near native hardware speed, and demonstrated system-level scale-out simulation studies using Olio Web 2.0 and Hadoop workloads.

5.10.2 Synthetic workload generation

This work is also inspired by earlier work in synthetic workload generation. Some prior work focuses on synthetic workloads for evaluating storage systems, see for example [24]. Other prior work focuses on generating

synthetic workloads and clones of general-purpose and embedded applications, such as SPEC CPU and MiBench, however, this work was limited to mostly CPU-centric workloads and systems, and did not consider more complex workloads that involve significant full-system activity as is the case for the database workload considered in this study. In particular, statistical simulation [18] [58] [60] collects program characteristics from a program execution and subsequently generates a synthetic trace from it that is then simulated on a simple, statistical trace-driven processor simulator. The important advantage of statistical simulation is twofold. Because the synthetic traces are very short, typically a few millions of instructions at most, they can be simulated in a short amount of time, making statistical simulation a valuable approach for early design stage exploration. Second, a synthetic trace hides proprietary information and does not leak business nor user-sensitive information to third parties. More recent work proposed automated synthetic benchmark generation [8] [33] [37] [74] which builds on the statistical simulation approach but generates a synthetic benchmark rather than a synthetic trace, which allows for running the synthetic workload on real hardware as well as execution-driven simulators.

Whereas statistical simulation and synthetic benchmark generation is a bottom-up approach, i.e., a workload is generated from specifying its behavioral characteristics, [73] proposed code mutation which hides proprietary information in a top-down manner. They mutate an existing benchmark so that reverse engineering the benchmark gets more complicated, while preserving similar execution behavior. Code obfuscation [15] is a related technique but converts a program into an equivalent program that is more difficult to understand and reverse engineer. There is a fundamental difference between code obfuscation and benchmark synthesis though. The goal of program obfuscation is to generate a transformed program that is functionally equivalent to the original program, i.e., when given the same input, the transformed program should produce the same output as the original program. The performance characteristics of the transformed program can be — and in practice they are — very different from the original program. Benchmark synthesis on the other hand generates a synthetic program that exhibits the same performance characteristics as the original program, however, its functionality can be very different.

5.10.3 Database performance analysis

Shao et al. [70] present DBmbench which is a microbenchmark suite for emulating Decision Support Systems (DSS) and Online Transaction Processing (OLTP) workloads at the computer architectural level. DBmbench includes two tables and three simple queries, and its key design principle is to keep the schemas and queries as simple as possible, and to focus on the dom-

inant operations in DSS and OLTP workloads. Our goal and approach is different. We aim at evaluating DBMS at the system level, and consider real-life database schemes and queries while hiding business and user information.

Morfonios et al. [54] present the approach used in Oracle 11g for accurately capturing database requests from multiple concurrent users in a production database environment. The recorded requests are then replayed in a test database environment. They pay special attention to synchronization to enforce specific orderings between the replayed requests. This enables accurately capturing timing and concurrency between database requests in a test environment. This approach assumes a test database environment that is identical to the production environment, which might not always be possible in practice because of privacy issues and/or because it requires taking an exact copy of the production database in the test environment.

There is significant interest in synthetic database generation in the database community. However, none of this prior work actually validates the synthetically generated databases against a real-life commercial database workload like we do in this work — several papers compare the synthetic database workload against TPC-C or TPC-H though. [72] argues that the handful of TPC benchmarks is becoming increasingly irrelevant to the multitude of data-centric applications observed these days. Instead, Tay advocates for techniques to synthetically scale up and down empirical data sets to match specific database applications. [14] propose a framework for generating synthetic databases with specific data distributions and query workloads; [31] use a graph-based database model; [27] are concerned with quickly generating very large databases. [12] generate synthetic databases for testing purposes, not for performance analysis.

5.11 Conclusion

In this chapter, we presented a framework for cloning database workloads. This is done by collecting a statistical profile of the original database, from which a synthetic database clone is then generated. The synthetic database clone anonymizes and obfuscates the original database so that performance analyses can be done by third parties without worrying about intellectual property and privacy issues. Synthetically generated database workloads allow for repeatable experiments in an offline environment that are representative for the original workload, and answering what-if questions by simply changing the statistical profile and generating a synthetic database workload from it.

We evaluated the proposed framework for a real-life database taken from the Netlog Web 2.0 workload, a popular social networking site in

Europe. We validated accuracy — the average response time is estimated with a 0.95% error, and the 90 percentile with a 1.38% error, using the synthetic workload compared to the original workload — and we demonstrated the usefulness of database cloning through various case studies in which we change both hardware and software configurations. We explored the impact of CPU clock frequency, hardware prefetching, hard drives versus solid-state disks, database scaling, alternative storage engines, and database query cache sizing. These case studies illustrated the ability to accurately mimic the original database workload behavior through cloning, and explore various design alternatives and trade-offs.

Chapter 6

Conclusions and Future Work

In this dissertation, we focused on optimizing the total cost of ownership of the data center. Because of the wide variety of data center applications, we also focused on characterizing which type of applications are running in the data center, and their impact on TCO. In this chapter, we first summarize the challenges, after which we draw conclusions for the presented research work. Finally, we present some ideas for future research.

6.1 Summary

The fast growth of the World Wide Web has introduced many challenges for service providers. One of the most important challenges is to reduce total cost of ownership. There are various factors affecting the cost of a data center, such as the hardware infrastructure (servers, racks and switches), power and cooling infrastructure, operating expenditure, and real estate. Hence, data centers are very cost-sensitive and need to be optimized for the ensemble.

The end-user is playing a central role in the expansion of the World Wide Web. Today's end-users have ubiquitous Internet access possibilities such as smartphones and tablets to access content that is residing on the Internet. Along with optimizing the total cost of ownership, it is also of great importance to minimize end-user latency. It is known from previous research that page loading times have direct influence on company profit [42].

In the remainder of this chapter, we briefly highlight the major findings and contributions of this dissertation.

6.1.1 Power-hungry workloads

We started this dissertation by tackling the traditional technique for dimensioning data centers. Common practice is to (over-)provision the power and cooling infrastructure of a data center by using nameplate power consumption, as provided by hardware vendors. As a first step, we presented a better approach for dimensioning the power and cooling grid by using realistic maximum power consumption. We presented a solution to the challenge of determining realistic maximum power consumption through a methodology for automatically generating so-called power viruses.

The proposed framework automatically generates full-system multi-core powermarks, or synthetic programs with desired power characteristics on multi-core server platforms. This framework was used to study the impact on the total energy cost of dimensioning the power supply units at the power usage determined by a max powermark rather than the nameplate power consumption. The powermarks exceed power consumption of existing performance benchmarks and torture tests by a significant margin. These powermarks can be used for constructing full-system power models that are reasonably accurate, easy to develop and use, and provide error bounds; in addition, the powermarks can be used for dimensioning power provisioning in server and data center infrastructures.

6.1.2 Data-centric workloads

Data explosion and diversity in the Internet drives the emergence of a new set of data-centric workloads to manage, manipulate, mine, index, compress, encrypt, etc. huge amounts of data, often referred to as 'big data'. In addition, the data is increasingly rich media, and includes images, audio and video, in addition to text. Given that the data centers hosting the online data and running these data-centric workloads are very much cost driven, it is important to understand how this emerging class of applications affects some of the design decisions in the data center.

Through the architectural simulation of minutes of run time of a set of data-centric workloads on a validated full-system x86 simulator, we derived the insight that high-end servers are more performance-cost efficient compared to commodity and low-end embedded servers for some workloads; for others, the low-end server or the commodity server is more performance-cost efficient. This suggests heterogeneous data centers as the optimum data center configuration. We conclude that the benefit from heterogeneity is very much workload, server-cost and electricity-cost dependent, and, for a specific setup, we report improvements up to 88%, 24% and 17% over a homogeneous high-end, commodity and low-end server data center, respectively. We also identify the sweet spot for heterogeneity

as a function of high-end versus low-end server cost, and we provide the insight that the benefit from heterogeneity increases at lower energy costs.

6.1.3 Web 2.0 workload characterization

Designing data centers for Web 2.0 social networking applications is a major challenge because of the large number of users, the large scale of the data centers, the distributed application base, and the cost sensitivity of a data center facility. Optimizing the data center for performance per dollar is far from trivial.

In this dissertation, we presented a case study in which we characterized a real-life Web 2.0 workload called Netlog, a popular social networking site in Europe, and evaluated hardware and software design choices in the data center. Our methodology samples the Web 2.0 workload both in space and in time to obtain a reduced workload that can be replayed, driven by input data captured from a real data center. The reduced workload captures the important services (and their interactions) and allows for evaluating how hardware and software choices affect end-user experience (response times).

We explored hardware trade-offs for the Web 2.0 workload in terms of core count, clock frequency, HDD versus SSD, etc., for the Web, memcached and database servers, and we obtain several interesting insights, such as the Web servers scale well with core count, and end-user response times are inversely proportional to Web server CPU frequency; an SSD reduces the longest response times by around 30% over an HDD in the database servers, which may or may not be justifiable given the significantly higher cost for SSD versus; memcached servers show low levels of CPU utilization, and are both memory and network-bound, hence, hardware choice should be driven by the cost of integrating more main memory in the server. Further, we presented two case studies illustrating how the method can be used for guiding hardware purchasing decisions as well as software optimizations.

6.1.4 Synthetic database cloning

The database management system is an important component of a contemporary Web 2.0 workload, yet improving its performance is challenging. Evaluating hardware and software alternatives and trade-offs in a production environment is complicated and might not always be possible; copying (part of) a database to an offline environment might not be feasible either, particularly because of intellectual property and privacy issues.

In this dissertation, we presented a framework for cloning database workloads. This is done by collecting a statistical profile of the original

database, from which a synthetic database clone is then generated. The synthetic database clone anonymizes and obfuscates the original database so that performance analyses can be done by third parties without worrying about intellectual property and privacy issues. Synthetically generated database workloads allow for repeatable experiments in an offline environment that are representative for the original workload, and answering what-if questions by simply changing the statistical profile and generating a synthetic database workload from it.

We evaluated the proposed framework for a real-life database taken from the Netlog Web 2.0 workload. We validated accuracy — the average response time is estimated with a 0.95% error, and the 90 percentile with a 1.38% error, using the synthetic workload compared to the original workload — and we demonstrated the usefulness of database cloning through various case studies in which we change both hardware and software configurations. We explored the impact of CPU clock frequency, hardware prefetching, hard drives versus solid-state disks, database scaling, alternative storage engines, and database query cache sizing. These case studies illustrated the ability to accurately mimic the original database workload behavior through cloning, and explore various design alternatives and trade-offs.

6.2 Future Work

Since the high-performance computing server market is still expected to grow 8% yearly until 2015, there will still be several research opportunities to further optimize the data center, with a special emphasis on power and TCO.[1] In the following sections we present some ideas for future work.

6.2.1 Power and energy efficiency

One of the key issues in supporting the growth of the World Wide Web is energy provisioning. We want data centers to be as energy-efficient as possible. In Chapter 2 we presented a technique to dimension the data center based on realistic maximum power consumption. In the presented results we see that energy proportionality is poor for current server platforms, i.e., some platforms still use more than 100 Watt when the CPU is idle.

CPUs have become more energy-efficient over the years, however memory, storage, cooling and other components did not evolve at the same pace. Work has to be done to reduce the static power consumed by these components. The benefit of improving energy proportionality of

[1] http://www.idc.com/getdoc.jsp?containerId=prUS23386912

future server platforms can potentially save huge amounts of energy, thus reducing TCO.

6.2.2 Workload cloning

In Chapter 4 we presented our work of cloning a real-life Web 2.0 workload. Setting up an entire workload, including web servers, caching servers and database servers in a separate testing environment is a tedious task. Furthermore, when replicating a proprietary Web application, web pages, databases, etc. have to be anonymized to protect company and user-sensitive information.

To speed up the task of replicating and anonymizing an entire database, we presented a technique to automatically build a synthetic database clone in Chapter 5. As part of future work, it would be of great interest to also make a synthetic duplicate of the other components of a real-life Web 2.0 workload. The ultimate goal could be to have an automated tool that automatically builds a synthetic clone of an entire Web 2.0 application. The first step in this tool would be to capture live incoming requests. These requests will need to be anonymized to protect company and user-sensitive information. Using this anonymization scheme, the tool would need to construct a synthetic duplicate of all web pages and a synthetic database clone. This would enable us to rebuild an entire Web application in an automated way.

6.2.3 Analytical workload modeling

In Chapter 4, we duplicated the entire Netlog workload from a production to a test environment in order to analyze and understand its performance characteristics. An alternative approach might be to construct an analytical model that describes the workload's key performance characteristics with inputs taken from hardware performance counters and system-level monitoring. The analytical model might then be helpful in guiding software and hardware optimizations, finding performance bottlenecks, and exploring what-if questions.

Building an analytical performance model for a commercial data center workload like Netlog would, without any doubt, be very challenging and it is unclear whether it would be possible to construct a white-box model from first principles, or whether one would need to resort to a black-box, empirical model, such as a neural network or regression model. An additional challenge, next to the model itself, is to collect the appropriate set of characteristics that serve as input to the model. The overhead for collecting these inputs would need to be zero (or at least very small) in order not to affect the production environment.

6.2.4 New workloads: Web 3.0?

In this dissertation we focused on different workloads running in the data center, several data-centric workloads, and a real-life Web 2.0 workload. Some experts believe that the next phase in the development of the World Wide Web is getting closer. Some call it Web 3.0, the web of data, or the semantic Web.[2,3] Web 3.0 technology is about connecting everything together in a semantic way. This enables not only humans, but also devices, to interpret information and respond to complex human requests, based on their meaning.

With the emergence of Web 3.0 technology, a new set of data center workloads will become available. It is important that the research community puts effort in understanding and characterizing these new workloads. Some of these workloads might have very different architectural requirements compared to their predecessors.

[2]http://www.tweakandtrick.com/2012/05/web-30.html
[3]http://en.wikipedia.org/wiki/Semantic_Web

Bibliography

[1] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A fast array of wimpy nodes. In *Proceedings of the International Symposium on Operating Systems Principles (SOSP)*, pages 1–14, October 2009.

[2] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega. COTSon: Infrastructure for full system simulation. *SIGOPS Operating System Review*, 43(1):52–61, January 2009.

[3] A. Avetisyan, R. Campbell, I. Gupta, M. Heath, S. Ko, G. Ganger, M. Kozuch, D. OHallaron, M. Kunze, T. Kwan, K. Lai, M. Lyons, D. Milojicic, H. Y. Lee, Y. C. Soh, N. K. Ming, J. Y. Luke, and H. H. Namgoong. Open cirrus: A global cloud computing testbed. *IEEE Computer*, 43(4):42–50, April 2010.

[4] D. A. Bader, Y. Li, T. Li, and V. Sachdeva. BioPerf: A benchmark suite to evaluate high-performance computer architecture on bioinformatics applications. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*, pages 163–173, October 2005.

[5] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, March 2003.

[6] L. A. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. Morgan and Claypool Publishers, 2009.

[7] R. Bedichek. SimNow: Fast platform simulation purely in software. In *Proceedings of the Symposium on High Performance Chips (HOT CHIPS)*, August 2004.

[8] R. Bell, Jr. and L. K. John. Improved automatic testcase synthesis for performance model validation. In *Proceedings of the 19th ACM International Conference on Supercomputing (ICS)*, pages 111–120, June 2005.

[9] R. Bertran, A. Buyuktosunoglu, M.S. Gupta, M. Gonzalez, and P. Bose. Systematic energy characterization of CMP / SMT processor systems

via automated micro-benchmarks. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 199–211, December 2012.

[10] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 72–81, October 2008.

[11] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *Computer Architecture News*, 39:1–7, May 2011.

[12] C. Binnig, D. Kossmann, E. Lo, and T. Ozsu. QAGen: Generating query-aware test databases. In *Proceedings of the ACM SIGMOD Internation Conference on Management of Data*, pages 341–352, June 2007.

[13] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA)*, pages 83–94, June 2000.

[14] N. Bruno and S. Chaudhuri. Flexible database generators. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1097–1107, August 2005.

[15] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical Report 148, The University of Auckland, July 1997.

[16] T. M. Conte, M. A. Hirsch, and K. N. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In *Proceedings of the International Conference on Computer Design (ICCD)*, pages 468–477, October 1996.

[17] C. Dirik and B. Jacob. The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 279–289, June 2009.

[18] L. Eeckhout, R. H. Bell Jr., B. Stougie, K. De Bosschere, and L. K. John. Control flow modeling in statistical simulation for accurate and efficient processor design studies. In *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA)*, pages 350–361, June 2004.

[19] L. Eeckhout, S. Nussbaum, J. E. Smith, and K. De Bosschere. Statistical simulation: Adding efficiency to the computer designer's toolbox. *IEEE Micro*, 23(5):26–38, Sept/Oct 2003.

[20] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 13–23, June 2007.

[21] W. Felter and T. Keller. Power measurement on the Apple Power Mac G5. Technical Report RC23276, IBM, 2004.

[22] M. Ferdman, A. Abileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 37–48, March 2012.

[23] K. Ganesan, J. Jo, W. L. Bircher, D. Kaseridis, Z. Yu, and L. K. John. System-level max power (SYMPO) — a systematic approach for escalating system level power consumption using synthetic benchmarks. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 19–28, September 2010.

[24] G. R. Ganger. Generating representative synthetic workloads: An unsolved problem. In *Proceedings of the Computer Measurement Group (CMG) Conference*, pages 1263–1269, December 1995.

[25] M. K. Gowan, L. L. Biro, and D. B. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. In *Proceedings of the 35th Design Automation Conference (DAC)*, pages 726–731, June 1998.

[26] J. Gray and B. Fitzgerald. Flash disk opportunity for server applications. *ACM Queue*, 6:18–23, July/August 2008.

[27] J. Gray, O. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger. Quickly generating billion-record synthetic databases. In *Proceedings of the ACM SIGMOD Internation Conference on Management of Data*, pages 243–252, May 1994.

[28] B. Grot, D. Hardy, P. Lotfi-Kamran, B. Falsafi, C. Nicopoulos, and Y. Sazeides. Optimizing data-center TCO with scale-out processors. *IEEE Micro*, 32(5):52–63, 2012.

[29] J. Hamilton. Datacenter networks are in my way. Principles of Amazon, October 2010. http://perspectives.mvdirona.com/.

[30] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Dynamic knobs for responsive power-aware computing. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 199–212, August 2011.

[31] K. Houkjaer, K. Torp, and R. Wind. Simple and realistic data generation. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1243–1246, September 2006.

[32] M. Huangs, P. Sallee, and M. Farrens. Branch transition rate: A new metric for improved branch claassification analysis. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pages 241–150, January 2000.

[33] C. Hughes and T. Li. Accelerating multi-core processor design space evaluation using automatic multi-threaded workload synthesis. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, pages 163–172, September 2008.

[34] Intel. *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Intel Corp., April 2012.

[35] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th Annual International Symposium on Microarchitecture (MICRO)*, pages 93–104, December 2003.

[36] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, fifth edition, 2002.

[37] A. M. Joshi, L. Eeckhout, R. Bell, Jr., and L. K. John. Distilling the essence of proprietary workloads into miniature benchmarks. *ACM Transactions on Architecture and Code Optimization (TACO)*, 5(2), August 2008.

[38] A. M. Joshi, L. Eeckhout, L. K. John, and C. Isen. Automated microprocessor stressmark generation. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pages 229–239, February 2008.

[39] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker. Performance characterization of a quad Pentium Pro SMP using OLTP workloads. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 15–26, June 1998.

[40] L. Keys, S. Rivoire, and J. D. Davis. The search for energy-efficient building blocks for the data center. In *The Second Workshop on Energy-Efficient Design (WEED), held in conjunction with the International Symposium on Computer Architecture (ISCA)*, June 2010.

[41] Y. Kim, L. K. John, S. Pant, S. Manne, M. Schulte, W. L. Bircher, and M. S. Sibi Govindan. AUDIT: Stress testing the automatic way. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 212–223, December 2012.

[42] R. Kohavi and R. Longbotham. Online experiments: Lessons learned. *IEEE Computer*, 40:85–87, September 2007.

[43] C. Kozyrakis, A. Kansal, S. Sankar, and K. Vaid. Server engineering insights for large-scale online services. *IEEE Micro*, 30:8–19, July/August 2010.

[44] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 81–92, December 2003.

[45] K.-D. Lange. Identifying shades of green: The SPECpower benchmarks. *IEEE Computer*, 42(3):95–97, March 2009.

[46] T. J. Lehman and M. J. Carey. A study of index structures for main memory database management systems. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 294–303, August 1986.

[47] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 315–326, June 2008.

[48] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi. Scale-out processors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 500–511, June 2012.

[49] Y. Luo, J. Rubio, L. K. John, P. Seshadri, and A. Mericas. Benchmarking internet servers on superscalar machines. *IEEE Computer*, 36(2):34–40, February 2003.

[50] J. Mars, L. Tang, and R. Hundt. Heterogeneity in 'homogeneous' warehouse-scale computers: A performance opportunity. *IEEE Comput. Archit. Lett.*, 10(2):29–32, July 2011.

[51] D. Meisner, B. T. Gold, and T. Wenisch. PowerNap: Eliminating server idle power. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating ystems (ASPLOS)*, pages 205–216, March 2009.

[52] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 319–330, June 2011.

[53] D. Meisner and T. Wenisch. Peak power modeling for data center servers with switched-mode power supplies. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 319–324, New York, NY, USA, 2010. ACM.

[54] K. Morfonios, R. Colle, L. Galanis, S. Buranawatanachoke, B. Dageville, K. Dias, and Y. Wang. Consistent synchronization schemes for workload replay. *Proceedings of the Very Large Data Bases (VLDB) Endowment*, 4:1225–1236, August 2011.

[55] T. Mudge and U. Hölzle. Challenges and opportunities for extremely energy-efficient processors. *IEEE Micro*, 30(4):20–24, July 2010.

[56] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, A. Choudhary, and J. Pisharath. MineBench: A benchmark suite for data mining workloads. In *Proceedings of the International Symposium on Computer Architecture (IISWC)*, pages 182–188, October 2006.

[57] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 182–188, October 2007.

[58] S. Nussbaum and J. E. Smith. Modeling superscalar processors via statistical simulation. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 15–24, September 2001.

[59] K. Olukotun, J. Laudon, and B. Lee. Mega-servers versus micro-blades for datacenter workloads. Panel debate at the Workshop on Architectural Concerns in Large Datacenters (ACLD), held with ISCA, June 2010.

[60] M. Oskin, F. T. Chong, and M. Farrens. HLS: Combining statistical and symbolic simulation to guide microprocessor design. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA)*, pages 71–82, June 2000.

[61] S. Polfliet, F. Ryckbosch, and L. Eeckhout. Studying hardware and software trade-offs for a real-life Web 2.0 workload. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE)*, pages 181–192, April 2012.

[62] Racktivity. *Racktivity RC0816*. Racktivity Inc., 2009. http://confluence.aserver.com/display/wwwrack/Smart+PDUs.

[63] P. Ranganathan. Green clouds and black swans in the exascale era. Keynote at the IEEE International Symposium on Workload Characterization (IISWC), October 2009.

[64] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. A. Barroso. Performance of database workloads on shared-memory systems with out-of-order processors. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 307–318, October 1998.

[65] V. J. Reddi, B. C. Lee, T. Chilimbi, and K. Vaid. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 26–36, June 2010.

[66] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. In *Proceedings of the Workshop on Power Aware Computing and Systems (HotPower), held at the Symposium on Operating Systems Design and Implementation (OSDI)*, December 2008.

[67] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: A balanced energy-efficiency benchmark. In *Proceedings of the SIGMOD International Conference on Management of Data*, pages 365–374, June 2007.

[68] F. Ryckbosch, S. Polfliet, and L. Eeckhout. Fast, accurate and validated full-system software simulation of x86 hardware. *IEEE Micro*, 30(6):46–56, Nov/Dec 2010.

[69] F. Ryckbosch, S. Polfliet, and L. Eeckhout. VSim: Simulating multi-server setups at near native hardware speed. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(4), January 2012.

[70] M. Shao, A. Ailamaki, and B. Falsafi. DBmbench: Fast and accurate database workload representation on modern microarchitecture. In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*, pages 254–267, October 2005.

[71] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 45–57, October 2002.

[72] Y. C. Tay. Data generation for application-specific benchmarking. *Proceedings of the Very Large Data Bases (VLDB) Endowment*, 4:1470–1473, August 2011.

[73] L. Van Ertvelde and L. Eeckhout. Dispersing proprietary applications as benchmarks through code mutation. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 201–210, March 2008.

[74] L. Van Ertvelde and L. Eeckhout. Benchmark synthesis for architecture and compiler exploration. In *The IEEE International Symposium on Workload Characterization (IISWC)*, pages 106–116, December 2010.

[75] V. Vasudevan, D. Andersen, M. Kaminsky, L. Tan, J. Franklin, and I. Moraru. Energy-efficient cluster computing with FAWN: Workloads and implications. In *Proceedings of the International Conference on Energy-Efficient Computing and Networking (e-Energy)*, pages 195–204, April 2010.

[76] R. Vishmanath, V. Wakharkar, A. Watwe, and V. Lebonheur. Thermal performance challenges from silicon to systems. *Intel Technology Journal*, 4(3), August 2000.

[77] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. SimFlex: Statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, July 2006.

[78] D. Wong and M. Annavaram. KnightShift: Scaling the energy proportionality wall through server-level heterogeneity. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 119–130, December 2012.

[79] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 84–95, June 2003.

[80] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of SimplePower: A cycle-accurate energy estimation tool. In *Proceedings of the 37th Design Automation Conference*, pages 340–345, June 2000.

[81] H. Zhao, I. Proctor, M. Yang, X. Qi, M. Williams, Q. Gao, G. Ottoni, A. Paroski, S. MacVicar, J. Evans, and S. Tu. The HipHop compiler for PHP. In *Proceedings of the International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 575–586, October 2012.