Improving Efficiency of Multicore in Warehouse Scale Computers

Jason Mars
Assistant Professor
University of Michigan, Ann Arbor
jasonmars.org
Landscape of Computing is Changing
Landscape of Computing is Changing
Landscape of Computing is Changing

"...the cloud was a $40.7 billion market in 2011 and will grow to $241 billion by 2020." -Forrester Research
Landscape of Computing is Changing
Modern Warehouse Scale Computers
Modern Warehouse Scale Computers
Modern Warehouse Scale Computers

- Large-scale internet applications

![Diagram of various internet applications]
Modern Warehouse Scale Computers

- Expensive - 100s of millions to billions of $$ to construct and operate

- Large-scale internet applications

- examples:
  - mail
  - social
  - search
Modern Warehouse Scale Computers

- Expensive - 100s of millions to billions of $$ to construct and operate
- Inefficient, as system and software architecture still in infancy

- Large-scale internet applications

Applications: mail, social, search
A Grand Challenge of WSCs
A Grand Challenge of WSCs

- Reliability
- Fault Tolerance
- Energy/Power
- Improve Efficiency
- Hardware Cost
- Performance
- Utilization
A Grand Challenge of WSCs

“...ultimately, software performance and server utilization matter just as much [as hardware costs].”

-Luiz Barroso and Urs Hoelzle
Figure 5.5: Activity profile of a sample of 5,000 Google servers over a period of 6 months.

When search traffic is high, all servers are being heavily used, but during periods of low traffic, a server might still see hundreds of queries per second, meaning that any idleness periods are likely to be no longer than a few milliseconds.

The absence of significant idle intervals despite the existence of periods of low activity is largely a result of applying sound design principles to high-performance, robust distributed systems software. Large-scale Internet services rely on efficient load distribution to a large number of servers, creating a situation where when load is lighter we tend to have a lower load in multiple servers instead of concentrating the load in fewer servers and idling the remaining ones. Idleness can be manufactured by the application (or an underlying cluster management system) by migrating workloads and their corresponding state to fewer machines during periods of low activity. This can be relatively easy to accomplish when using simple replication models, when servers are mostly stateless (i.e., serving data that resides on a shared NAS or SAN storage system). However, it comes at a cost in terms of software complexity and energy for more complex data distribution models or those with significant state and aggressive exploitation of data locality.

Another reason why it may be difficult to manufacture useful idle periods in large-scale distributed systems is the need for resilient distributed storage. GFS, the Google File System, achieves higher resilience by distributing data chunk replicas for a given file across an entire cluster instead of concentrating them within only a small number of machines. This benefits file system performance by achieving fine granularity load balancing, as well as resiliency, because when a storage server

A Grand Challenge of WSCs

Improve Efficiency

Performance

Utilization

Fraction of Time

Utilization

Tuesday, October 8, 13
FIGURE 5.5: Activity profile of a sample of 5,000 Google servers over a period of 6 months.

When search traffic is high, all servers are being heavily used, but during periods of low traffic, a server might still see hundreds of queries per second, meaning that any idleness periods are likely to be no longer than a few milliseconds.

The absence of significant idle intervals despite the existence of periods of low activity is largely a result of applying sound design principles to high-performance, robust distributed systems software. Large-scale Internet services rely on efficient load distribution to a large number of servers, creating a situation where when load is lighter we tend to have a lower load in multiple servers instead of concentrating the load in fewer servers and idling the remaining ones. Idleness can be manufactured by the application (or an underlying cluster management system) by migrating workloads and their corresponding state to fewer machines during periods of low activity. This can be relatively easy to accomplish when using simple replication models, when servers are mostly stateless (i.e., serving data that resides on a shared NAS or SAN storage system). However, it comes at a cost in terms of software complexity and energy for more complex data distribution models or those with significant state and aggressive exploitation of data locality.

Another reason why it may be difficult to manufacture useful idle periods in large-scale distributed systems is the need for resilient distributed storage. GFS \[31\], the Google File System, achieves higher resilience by distributing data chunk replicas for a given file across an entire cluster instead of concentrating them within only a small number of machines. This benefits file system performance by achieving fine granularity load balancing, as well as resiliency, because when a storage server...
A Grand Challenge of WSCs

- 1% improvement results in millions $$ saved
Reflecting on WSC Design
Reflecting on WSC Design

• Design philosophy of WSCs
Reflecting on WSC Design

- Design philosophy of WSCs

Functionality First  
Efficiency Second
Reflecting on WSC Design

• Design philosophy of WSCs

  Functionality First

  Efficiency Second

• As a result...
  • Use commodity components, multicore processors (x86), open source (linux, gcc), etc.
  • Stitched together for functionality
  • Then tweaked for better efficiency
Reflecting on WSC Design

• Design philosophy of WSCs

Functionality First

Efficiency Second

• As a result...
  • Use commodity components, multicore processors (x86), open source (linux, gcc), etc.
  • Stitched together for functionality
  • Then tweaked for better efficiency

• Problem
  • Commodity components not designed for WSC
  • Starting with commodity components, lose sight of unique characteristics of WSCs -> Inefficient design!
Current View of Underlying Hardware

• Underlying hardware architecture and microarchitecture more nuanced
Current View of Underlying Hardware

- Underlying hardware architecture and microarchitecture more nuanced
Current View of Underlying Hardware

- Underlying hardware architecture and microarchitecture more nuanced
Multicore in WSC

Socket 1

Socket 2

L3

QP/HT (Mem CTRL)

L2

L1

Core

L1

L1

L1

Core

Core

Core

Tread A
Tread B

0 8 1 9 2 10 3 11

4 12 5 13 6 14 7 15

L2

L3

L2

QP/HT (Mem CTRL)
Multicore in WSC

Socket 1

0  8  1  9  2  10  3  11

Thread A  Thread B

Core Core Core

L1 L1 L1

L2 L2

L3

QP/HT (Mem CTRL)

Socket 2

4  12  5  13  6  14  7  15

Core Core Core Core

L1 L1 L1 L1

L2 L2

L3

QP/HT (Mem CTRL)
Multicore in WSC

Socket 1

Socket 2

Tread A

Tread B

Core

Core

Core

Core

L1

L1

L1

L1

L2

L2

L2

L2

QP/HT (Mem CTRL)

QP/HT (Mem CTRL)
Multicore in WSC

Socket 1

- Tread A
  - Core
  - L1
  - L2
  - L3
  - QP/HT (Mem CTRL)

Socket 2

- Tread B
  - Core
  - L1
  - L2
  - L3
  - QP/HT (Mem CTRL)
Multicore in WSC
Multicore in WSC

- Software stack needs to be aware of underlying multicore microarchitecture
- And interactions with application characteristics
Reflecting on WSC Design

• Important to acknowledge key characteristics that steer design
Reflecting on WSC Design

• Important to acknowledge key characteristics that steer design

• We must rethink the system architecture!
Rethinking WSC Architecture
Rethinking WSC Architecture

• Observation: Key characteristic unique to WSCs has been overlooked
Rethinking WSC Architecture

• Observation: Key characteristic unique to WSCs has been overlooked
  • Diversity in Execution Environments (EE)
Rethinking WSC Architecture

• Observation: Key characteristic unique to WSCs has been overlooked
  • Diversity in Execution Environments (EE)

Definition: Machine configuration coupled with simultaneously co-running tasks

Gen 2 Xeon
1 Job
Rethinking WSC Architecture

- Observation: Key characteristic unique to WSCs has been overlooked
  - Diversity in Execution Environments (EE)
Rethinking WSC Architecture

- Observation: Key characteristic unique to WSCs has been overlooked
  - Diversity in Execution Environments (EE)
Rethinking WSC Architecture

- Observation: Key characteristic unique to WSCs has been overlooked
  - Diversity in Execution Environments (EE)

- Job Mapper
  - Gen 1 Xeon: 0 Jobs
  - Gen 2 Xeon: 1 Job
  - Opteron: 3 Jobs

- websearch job
- compression job
Rethinking WSC Architecture

• Observation: Key characteristic unique to WSCs has been overlooked
  • Diversity in Execution Environments (EE)

  ![Diagram showing job mapper and different execution environments](image)

  • Tasks aren’t placed where they run best

  - Gen 1 Xeon: 0 Jobs
  - Gen 2 Xeon: 1 Job
  - Opteron: 3 Jobs
  - websearch job
  - compression job
Rethinking WSC Architecture

• Observation: Key characteristic unique to WSCs has been overlooked

  • Diversity in Execution Environments (EE)

  - Tasks aren’t placed where they run best
  - Tasks can’t adapt to changes in environment
Rethinking WSC Architecture

- Observation: Key characteristic unique to WSCs has been overlooked
  - Diversity in Execution Environments (EE)

Tasks aren’t placed where they run best
- Tasks can’t adapt to changes in environment
- Can not manage interference between tasks within environment
Rethinking WSC Architecture

• Observation: Key characteristic unique to WSCs has been overlooked
  • Diversity in Execution Environments (EE)

  - Jobs aren’t placed where they run best
  - Tasks can’t adapt to changes in environment
  - Can not manage interference between tasks within environment

• Diversity in EE is key for a highly efficient design!
Observation: Impact of EE on Performance

Machine Configurations

- Gen 1 Xeon CLOVER
- Opteron ISTAN
- Gen 2 Xeon WEST
Observation: Impact of EE on Performance

Machine Configurations

- Gen 1 Xeon CLOVER
- Opteron ISTAN
- Gen 2 Xeon WEST

Figure 3.1: Performance comparison of key Google applications across three microarchitectures. Each cluster is normalized to poorest performing architecture (the higher the better). Table 6.2 also presents a description for each application. Each application corresponds to an actual binary that is run in the datacenter. These applications are part of a test infrastructure developed internally at Google composed of a host of Google workloads and machine clusters that have been laboriously configured by a team of engineers for performance analysis and optimization testing across Google. Each application shown in the table operates on a repeatable log of thousands of queries from actual user activity from production. We use this test infrastructure throughout the remainder of this work. The number of cores used by each application is configured to three for both solo and co-location runs.

3.1.2 Microarchitectural Heterogeneity

We first characterize the performance variability due to microarchitectural heterogeneity in WSCs. In addition to quantifying the magnitude of the performance variability, our study also aims to investigate firstly whether any microarchitecture consistently outperforms others for all applications; and secondly whether there is a difference in how sensitive the performance...
Observation: Impact of EE on Performance

Machine Configurations

- Gen 1 Xeon CLOVER
- Opteron ISTAN
- Gen 2 Xeon WEST

Figure 3.1: Performance comparison of key Google applications across three microarchitectures. Each cluster is normalized to poorest performing architecture (the higher the better).

<table>
<thead>
<tr>
<th>Machine Configurations</th>
<th>Normalized Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bigtable</td>
<td>3.5x</td>
</tr>
<tr>
<td>Ads-servlet</td>
<td>3x</td>
</tr>
<tr>
<td>Maps-detect-face</td>
<td>2.5x</td>
</tr>
<tr>
<td>Search-render</td>
<td>2x</td>
</tr>
<tr>
<td>Search-scoring</td>
<td>1.5x</td>
</tr>
<tr>
<td>Protobuf</td>
<td>1x</td>
</tr>
<tr>
<td>Docs-analyzer</td>
<td></td>
</tr>
<tr>
<td>Saw-countw</td>
<td></td>
</tr>
<tr>
<td>Youtube-x264yt</td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- Red: Clover
- Green: Istan
- Blue: West

Tuesday, October 8, 13
# Observation: Impact of EE on Performance

## Machine Configurations

<table>
<thead>
<tr>
<th>Machine Configuration</th>
<th>Gen 1 Xeon CLOVER</th>
<th>Opteron ISTAN</th>
<th>Gen 2 Xeon WEST</th>
</tr>
</thead>
</table>

## Figure 3.1: Performance comparison of key Google applications across three microarchitectures. Each cluster is normalized to poorest performing architecture (the higher the better).

<table>
<thead>
<tr>
<th>Application</th>
<th>Gen 1 Xeon CLOVER</th>
<th>Opteron ISTAN</th>
<th>Gen 2 Xeon WEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bigtable</td>
<td>3.5x</td>
<td>3x</td>
<td>3x</td>
</tr>
<tr>
<td>ads-servlet</td>
<td>2.5x</td>
<td>2x</td>
<td>2x</td>
</tr>
<tr>
<td>maps-detect-face</td>
<td>1.5x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td>search-render</td>
<td>1.5x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td>search-scoring</td>
<td>1.5x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td>protobuf</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td>docs-analyzer</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td>saw-countw</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td>youtube-x264yt</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td>Mean</td>
<td>1.5x</td>
<td>1x</td>
<td>1x</td>
</tr>
</tbody>
</table>

The applications are part of a test infrastructure developed internally at Google composed of a host of Google workloads and machine clusters that have been both laboriously configured by a team of engineers for performance analysis and optimization testing across Google. Each application shown in the table operates on a repeatable log of thousands of queries from actual user activity from production. We use this test infrastructure throughout the remainder of this work. The number of cores used by each application is configured to three for both solo and co-location runs.

## 3.1.2 Microarchitectural Heterogeneity

We first characterize the performance variability due to microarchitectural heterogeneity in WSCs. In addition to quantifying the magnitude of the performance variability, our study also aims to investigate firstly whether any microarchitecture consistently outperforms others for all applications; and secondly whether there is a difference in how sensitive the performance...
Observation: Impact of EE on Performance

Machine Configurations

![Machine Configurations Diagram]

Figure 3.1: Performance comparison of key Google applications across three microarchitectures. Each cluster is normalized to poorest performing architecture (the higher the better).

Table 6.2 also presents a description for each application. Each application corresponds to an actual binary that is run in the datacenter. These applications are part of a test infrastructure developed internally at Google composed of a host of Google workloads and machine clusters that have been both laboriously configured by a team of engineers for performance analysis and optimization testing across Google. Each application shown in the table operates on a repeatable log of thousands of queries from actual user activity from production. We use this test infrastructure throughout the remainder of this work. The number of cores used by each application is configured to three for both solo and co-location runs.

3.1.2 Microarchitectural Heterogeneity

We first characterize the performance variability due to microarchitectural heterogeneity in WSCs. In addition to quantifying the magnitude of the performance variability, our study also aims to investigate firstly whether any microarchitecture consistently outperforms others for all applications; and secondly whether there is a difference in how sensitive the performance...
Observation: Impact of EE on Performance

Machine Configurations

Task performance is heavily impacted by diverse machine configurations

Figure 3.1: Performance comparison of key Google applications across three microarchitectures. Each cluster is normalized to poorest performing architecture (the higher the better)

Table 6.2 also presents a description for each application. Each application corresponds to an actual binary that is run in the datacenter. The applications are part of a test infrastructure developed internally at Google composed of a host of Google workloads and machine clusters that have been laboriously configured by a team of engineers for performance analysis and optimization testing across Google. Each application shown in the table operates on a repeatable log of thousands of queries from actual user activity from production. We use this test infrastructure throughout the remainder of this work. The number of cores used by each application is configured to three for both solo and co-location runs.

3.1.2 Microarchitectural Heterogeneity

We first characterize the performance variability due to microarchitectural heterogeneity in WSCs. In addition to quantifying the magnitude of the performance variability, our study also aims to investigate firstly whether any microarchitecture consistently outperforms others for all applications; and secondly whether there is a difference in how sensitive the performance
Observation: Impact of EE on Performance

Machine Configurations

- Gen 1 Xeon CLOVER
- Opteron ISTAN
- Gen 2 Xeon WEST
Observation: Impact of EE on Performance

Machine Configurations
Observation: Impact of EE on Performance

Figure 3.1 presents the experimental results for our Google testbed with 9 key Google applications running on 3 types of production machines. The y-axis shows the performance (average instructions per second) of each application on three types of machines, normalized by the worst performance among the three for each application.

Machine Configurations

- **Gen 1 Xeon CLOVER**
- **Gen 2 Xeon WEST**
- **Opteron ISTAN**

Performance Impact

- **bigtable**: BT on Clover
- **ads-servlet**: BT on Istan
- **maps-detect-face**: BT on West
- **search-render**: SS on Clover
- **search-scoring**: SS on Istan
- **protobuf**: SS on West
- **docs-analyzer**: PB on Clover
- **saw-mars**: PB on Istan
- **youtube-x264yt**: PB on West

Figure 3.2: Google application performance when co-located with bigtable (BT), search-scoring (SS), and protobuf (PB). Negative indicates slowdown of each application is to varying platform types. As we will discuss later in this section, the variance in performance sensitivity across all applications in a workload is important for exploiting the heterogeneity.

Figure 3.1 shows that even among three architectures that are from competing generations, there is a significant performance variability for Google applications. More interestingly, no platform is consistently better than the others in this experiment. Although the Westmere Xeon outperforms the other platforms for most applications, maps-detect-face running on the Istanbul Opteron outperforms the Westmere Xeon by around 25%. On the other hand, the Clovertown Xeon and Istanbul Opteron compete much more closely. It is also important to note that even though the Westmere Xeon platform is almost always better than the other two, the performance sensitivity of each application varies.
Observation: Impact of EE on Performance

Figure 3.1 presents the experimental results for our Google testbed with 9 key Google applications running on 3 types of production machines. The y-axis shows the performance (average instructions per second) of each application on three types of machines, normalized by the worst performance among the three for each application.

Figure 3.2: Google application performance when co-located with bigtable (BT), search-scoring (SS), and protobuf (PB). Negative indicates slowdown of each application is to varying platform types. As we will discuss later this section, the variance in performance sensitivity across all applications in a workload is important for exploiting the heterogeneity.

Figure 3.1 shows that even among three architectures that are from competing generations, there is a significant performance variability for Google applications. More interestingly, no platform is consistently better than the others in this experiment. Although the Westmere Xeon outperforms the other platforms for most applications, maps-detect-face running on the Istanbul Opteron outperforms the Westmere Xeon by around 25%. On the other hand, the Clovertown Xeon and Istanbul Opteron compete much more closely. It is also important to note that even though the Westmere Xeon platform is almost always better than the other two, the performance sensitivity of some applications is more pronounced on other platforms.

Machine Configurations

- Gen 1 Xeon CLOVER
- Opteron ISTAN
- Gen 2 Xeon WEST

Co-runners

- bigtable
- ads-servlet
- maps-detect-face
- search-render
- search-scoring
- protobuf
- docs-analyzer
- saw-mir
- youtube-x264yt

Performance Impact

- BT on Clover
- BT on Istan
- BT on West
- SS on Clover
- SS on Istan
- SS on West
- PB on Clover
- PB on Istan
- PB on West

worse
Observation: Impact of EE on Performance

Machine Configurations

- Gen 1 Xeon CLOVER
- Opteron ISTAN
- Gen 2 Xeon WEST

Co-runners

Task performance is heavily impacted by diverse co-runners

Figure 3.2: Google application performance when co-located with bigtable (BT), search-scoring (SS), and protobuf (PB). Negative indicates slowdown of each application is to varying platform types. As we will discuss later in this section, the variance in performance sensitivity across all applications in a workload is important for exploiting the heterogeneity.

Figure 3.1 presents the experimental results for our Google testbed with 9 key Google applications running on 3 types of production machines. The y-axis shows the performance (average instructions per second) of each application on three types of machines, normalized by the worst performance among the three for each application.

Docs-analyzer's data on Istanbul is missing because it is not configured for that particular platform.

Figure 3.1 shows that even among three architectures that are from competing generations, there is a significant performance variability for Google applications. More interestingly, no platform is consistently better than the others in this experiment. Although the Westmere Xeon outperforms the other platforms for most applications, maps-detect-face running on the Istanbul Opteron outperforms the Westmere Xeon by around 25%. On the other hand, the Clover-town Xeon and Istanbul Opteron compete much more closely. It is also important to note that even though the Westmere Xeon platform is almost always better than the other two, the performance sensitivity is worse for most applications.
Observation: Impact of EE on Performance

Observation: The execution environment has a significant impact on performance.
Observation: The execution environment has a significant impact on performance.

Exploiting and adapting to diversity in EE are critical for improving performance and utilization of WSCs.
Work Across the Stack

Characterization / Workload Studies

- commercial WSC workloads
  - ISCA '11
- impact of thread mappings
  - ISPASS '12
- characterizing interference
  - HIPEAC '11

Compilers / Analyses

- compiling for niceness
  - CGO '12
- efficient binary translation
  - CGO '07

- ReQoS
  - ASPLOS '13
- SBO
  - CGO '09
- hardware/software testing
  - ICSE '11

Runtimes / System Software

- heterogeneous mapping
  - CAL '11
- Heterogeneity in WSCs
  - ISCA '13
- Bubble-Up
  - MICRO '11
- Bubble-PiPo
  - ISCA '13
- hardware software testing
  - TACO '11
- CAER
  - CGO '10
- Bubble-Up
  - IEEE MICRO '12

Computer Architecture

- MP Hybrid Architecture
  - ISCA '12
- Cross-layer Heterogeneity
  - MEMOCODE '09

Google - NUMA
- HPCA '13

Tuesday, October 8, 13
Work Across the Stack

Characterization / Workload Studies
- commercial WSC workloads
  ISCA '11
- impact of thread mappings
  ISPASS '12
- characterizing interference
  HIPEAC '11

Compilers / Analyses
- compiling for niceness
  CGO '12
- efficient binary translation
  CGO '07
- ReQoS
  ASPLOS '13
- SBO
  CGO '09

Runtimes / System Software
- heterogeneous mapping
  CAL '11
- hardware software testing
  TACO '11
- CAER
  CGO '10
- Bubble-Up
  MICRO '11
- Bubble-Up
  IEEE MICRO '12

Computer Architecture
- MP Hybrid Architecture
  ISCA '12
- Cross-layer Heterogeneity
  MEMOCODE '09

Google - NUMA
HPCA '13

Tuesday, October 8, 13
Three Key Design Points for EE Diversity

Job Mapper

Gen 1 Xeon
0 Jobs

Gen 2 Xeon
1 Job Running

Opteron
3 Jobs Running
Three Key Design Points for EE Diversity

- Three Design Points
  - Cluster level awareness of EE
  - Machine level awareness of EE
  - Interference at Cluster and Machine levels
Three Key Design Points for EE Diversity

• Three Design Points
  • Cluster level awareness of EE
  • Machine level awareness of EE
  • Interference at Cluster and Machine levels
Three Key Design Points for EE Diversity

- Three Design Points
  - Cluster level awareness of EE
  - Machine level awareness of EE
  - Interference at Cluster and Machine levels
Three Key Design Points for EE Diversity

- Three Design Points
  - Cluster level awareness of EE
  - Machine level awareness of EE
  - Interference at Cluster and Machine levels
Three Key Design Points for EE Diversity

- Three Design Points
  - Cluster level awareness of EE
  - Machine level awareness of EE
  - Interference at Cluster and Machine levels
Talk Outline

• Introduction ✓

• Adapting to EE diversity in WSCs
  • Interference at Cluster and Machine levels
    • Machine Level
    • Cluster Level

• Wrap Up
Talk Outline

• Introduction ✓

• Adapting to EE diversity in WSCs
  • Interference at Cluster and Machine levels →
    • Machine Level
    • Cluster Level

• Wrap Up
Interference at Cluster and Machine Levels

Job Mapper

Gen 1 Xeon
0 Jobs

Gen 2 Xeon
1 Job Running

Opteron
3 Jobs Running
Interference at Cluster and Machine Levels
The Challenge of Contention
The Challenge of Contention
The Challenge of Contention
The Challenge of Contention
The Challenge of Contention
The Challenge of Contention
The Challenge of Contention
Contention Leads to Low Utilization

High priority application

Low priority application
Contention Leads to Low Utilization

Option A: Disallow Colocation.
- Low utilization
- Peak performance and QoS
Contention Leads to Low Utilization

Option A: Disallow Colocation.
- Low utilization
- Peak performance and QoS

Option B: Allow Colocation.
- High utilization
- Significant performance/QoS degradation

High priority application
Low priority application
Contention Leads to Low Utilization

**Option A: Disallow Colocation.**
- Low utilization
- Peak performance and QoS

**Option B: Allow Colocation.**
- High utilization
- Significant performance/QoS degradation

Colocation often disallowed
Contention Leads to Low Utilization

Option A: Disallow Colocation.
- Low utilization
- Peak performance and QoS

Option B: Allow Colocation.
- High utilization
- Significant performance/QoS degradation

Colocation often disallowed
The Uncertainty of Interference
The Uncertainty of Interference

• Uncertainty across co-runners
The Uncertainty of Interference

• Uncertainty across co-runners

• We can allow some interference, but not too much
The Uncertainty of Interference

- Uncertainty across co-runners
- We can allow some interference, but not too much
The Uncertainty of Interference

- Uncertainty across co-runners
- We can allow some interference, but not too much

Quality of Service (QoS) threshold

Performance of Search Render when Co-located

- ads
- bigtable
- maps
- protobuf
- sawzall
- semantic

QoS (1/latency)

Figure 1: Some co-locations violate allowable performance interference. The co-location of some single socket. The horizontal line shows the maximum allowable performance interference across-cores, negatively and unpredictably impacts the utilization of the computing resources in the datacenter.
The Uncertainty of Interference

- Uncertainty across co-runners
- We can allow some interference, but not too much
Can We Eliminate Uncertainty?

- Eliminate uncertainty of interference penalty
  - Precisely predict impact on QoS
  - Given arbitrary co-locations, allow “safe” co-locations
Can We Eliminate Uncertainty?

• Eliminate uncertainty of interference penalty
  • Precisely predict impact on QoS
  • Given arbitrary co-locations, allow “safe” co-locations

• Goals
  • General methodology, platform agnostic
  • Deployable at the scale of warehouse scale computers
Challenges

• Prior work
  • Can predict \textit{whether} an application is contentious
  • but not \textit{how much} it will hurt neighbors
Challenges

• Prior work
  • Can predict *whether* an application is contentious
  • but not *how much* it will hurt neighbors
• Capture interaction with resources not explicitly manageable (cache / bandwidth)
Challenges

• Prior work
  • Can predict *whether* an application is contentious
  • but not *how much* it will hurt neighbors
• Capture interaction with resources not explicitly manageable (cache / bandwidth)
  • Quadratic (brute-force) profiling methodology straightforward
  • Not suitable at the scale of WSC
Challenges

• Prior work
  • Can predict *whether* an application is contentious
  • but not *how much* it will hurt neighbors
• Capture interaction with resources not explicitly manageable (cache / bandwidth)
• Quadratic (brute-force) profiling methodology straightforward
  • Not suitable at the scale of WSC
• Is a linear solution possible?
  • Especially considering the precise impact is based on co-runner
## Insights

<table>
<thead>
<tr>
<th>White Box Approach</th>
<th>Is there a Black Box Approach?</th>
</tr>
</thead>
</table>

Tuesday, October 8, 13
## Insights

<table>
<thead>
<tr>
<th>White Box Approach</th>
<th>Is there a Black Box Approach?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Insights

White Box Approach

- caches
- bandwidth to mem
- secret sauce
- bus / interconnect

| prefetchers | replacement algo |
| mem controller | queues / buffers |

Is there a Black Box Approach?
Insights

**White Box Approach**

- caches
- prefetchers
- bandwidth to mem
- bus / interconnect
- secret sauce

**Is there a Black Box Approach?**

- mem controller
- replacement algo
- queues / buffers

- High complexity
- Not portable
- May not be feasible
## Insights

<table>
<thead>
<tr>
<th>White Box Approach</th>
<th>Is there a Black Box Approach?</th>
</tr>
</thead>
<tbody>
<tr>
<td>caches</td>
<td></td>
</tr>
<tr>
<td>prefetchers</td>
<td></td>
</tr>
<tr>
<td>bandwidth to mem</td>
<td></td>
</tr>
<tr>
<td>secret sauce</td>
<td>Black Box</td>
</tr>
<tr>
<td>bus / interconnect</td>
<td></td>
</tr>
<tr>
<td>mem controller</td>
<td></td>
</tr>
<tr>
<td>replacement algo</td>
<td></td>
</tr>
<tr>
<td>queues / buffers</td>
<td></td>
</tr>
</tbody>
</table>

- High complexity
- Not portable
- May not be feasible
Insights

<table>
<thead>
<tr>
<th>White Box Approach</th>
<th>Is there a Black Box Approach?</th>
</tr>
</thead>
<tbody>
<tr>
<td>caches</td>
<td></td>
</tr>
<tr>
<td>prefetchers</td>
<td></td>
</tr>
<tr>
<td>bandwidth to mem</td>
<td></td>
</tr>
<tr>
<td>buses / interconnect</td>
<td>Black Box</td>
</tr>
<tr>
<td>secret sauce</td>
<td></td>
</tr>
<tr>
<td>mem controller</td>
<td></td>
</tr>
<tr>
<td>replacement algo</td>
<td></td>
</tr>
<tr>
<td>queues / buffers</td>
<td></td>
</tr>
</tbody>
</table>

• High complexity
• Not portable
• May not be feasible

• Lower complexity
• Portable
• Deployable
Insights

### White Box Approach

- **caches**
- **prefetchers**
- **bandwidth to mem**
- **secret sauce**
- **bus / interconnect**
- **mem controller**
- **replacement algo**
- **queues / buffers**

- High complexity
- Not portable
- May not be feasible

### Is there a Black Box Approach?

- Lower complexity
- Portable
- Deployable
- (Wo)man in a dark room

---

Tuesday, October 8, 13
Bubble-Up!
Bubble-Up!

• Capture *sensitivity* and *aggressiveness*
Bubble-Up!

- Capture **sensitivity** and **aggressiveness**

- When deciding a co-location...
  - Use representation of sensitivity of app
  - Use representation of aggressiveness of co-runner
  - Combine to produce prediction of app’s degradation
Bubble-Up!

Search Render  Co-Runner
Bubble-Up!

sensitivity

Search Render  Co-Runner
Bubble-Up!

In this section, we describe how large-scale web-services are run in modern datacenters. We then discuss QoS and performance interference of co-running applications can be decoupled from the underlying infrastructure.

We present the design of Bubble-Up, a two-step characterization process. First, we evaluate the prediction accuracy and the improvement in utilization when applying Bubble-Up.

In addition to demonstrating the prediction accuracy when co-located on production servers, we also characterize their propensity to performance interference.

We reveal a new family of machine learning-based tools to steer pairwise co-locations of Google applications. We evaluate the prediction accuracy and the improvement in machine utilization in the cluster by 50%–90%, depending on the latency-sensitive applications' allowable QoS threshold.

A metric reduces the complexity of co-location analysis. As an underlying hypothesis, both pressure and sensitivity can be allowed, resulting in improved utilization in locations that do not violate the QoS threshold of an application.

With this information, co-location prediction is one that provides an expected amount of performance degradation that results from contention.

While others violate the threshold (dark bars), an application performing at 90% of full performance.

The profiling complexity for all pair-wise co-locations is prohibitively expensive. The profiling complexity for all pair-wise co-locations is prohibitively expensive.

This is a challenging problem. The most relevant resources include the number of cores, amount of memory, that specifies the machine level resources required. These are run in modern datacenters.

We then discuss QoS and machine utilization in the cluster by 50%–90%, depending on the latency-sensitive applications' allowable QoS threshold.

In this section, we describe how large-scale web-services are run in modern datacenters. We then discuss QoS and performance interference of co-running applications can be decoupled from the underlying infrastructure.

We present the design of Bubble-Up, a two-step characterization process. First, we evaluate the prediction accuracy and the improvement in utilization when applying Bubble-Up.

In addition to demonstrating the prediction accuracy when co-located on production servers, we also characterize their propensity to performance interference.

We reveal a new family of machine learning-based tools to steer pairwise co-locations of Google applications. We evaluate the prediction accuracy and the improvement in machine utilization in the cluster by 50%–90%, depending on the latency-sensitive applications' allowable QoS threshold.

A metric reduces the complexity of co-location analysis. As an underlying hypothesis, both pressure and sensitivity can be allowed, resulting in improved utilization in locations that do not violate the QoS threshold of an application.

With this information, co-location prediction is one that provides an expected amount of performance degradation that results from contention.

While others violate the threshold (dark bars), an application performing at 90% of full performance.

The profiling complexity for all pair-wise co-locations is prohibitively expensive. The profiling complexity for all pair-wise co-locations is prohibitively expensive.

This is a challenging problem. The most relevant resources include the number of cores, amount of memory, that specifies the machine level resources required. These are run in modern datacenters.

We then discuss QoS and machine utilization in the cluster by 50%–90%, depending on the latency-sensitive applications' allowable QoS threshold.

In this section, we describe how large-scale web-services are run in modern datacenters. We then discuss QoS and performance interference of co-running applications can be decoupled from the underlying infrastructure.

We present the design of Bubble-Up, a two-step characterization process. First, we evaluate the prediction accuracy and the improvement in utilization when applying Bubble-Up.

In addition to demonstrating the prediction accuracy when co-located on production servers, we also characterize their propensity to performance interference.

We reveal a new family of machine learning-based tools to steer pairwise co-locations of Google applications. We evaluate the prediction accuracy and the improvement in machine utilization in the cluster by 50%–90%, depending on the latency-sensitive applications' allowable QoS threshold.
Bubble-Up!

Figure 2: Example sensitivity curve for Bubble-Up!

Search Render | Co-Runner
--- | ---
Core | Core
L1 | L1
L1 | L1
L2 | L2
Mem Controller

A core insight of Bubble-Up is that predicting the performance of an application is critical to effective co-location. This is because the performance of an application can be significantly affected by its co-locations. Therefore, it is important to predict the performance of an application before it is co-located with other applications.

In Bubble-Up, we use a two-step characterization process. First, we identify the bubble's pressure score for an application. If the bubble's pressure score is less than 1%, we can predict that the application will perform at less than 1% of its full performance. On the other hand, if the bubble's pressure score is greater than 1%, we can predict that the application will perform at 90% of its full performance. This allows us to predict the performance of an application with high accuracy.

In the second step of Bubble-Up, we identify the bubble's aggressiveness score. This score is used to determine how aggressively an application should be co-located. Applications with low aggressiveness scores should be co-located with other applications, while applications with high aggressiveness scores should be co-located with applications that have similar characteristics.

Overall, Bubble-Up is a powerful tool for predicting the performance of applications in co-location environments. It allows us to predict the performance of applications with high accuracy, and to co-locate applications in a way that maximizes performance and minimizes interference.
Bubble-Up!

Search Render
Core → Core
L1 → L1
L2

Co-Runner
Core → Core
L1 → L1

L2

Mem Controller

sensitivity

aggressiveness

2

app QoS

100%
90%
80%

bubble's pressure

2 ..... 10

The specific contributions of this work are as follows:

• We introduce 17 production Google workloads and characterize them.
• We present the design of Bubble-Up, a methodology to steer pairwise co-locations of Google applications.
• We evaluate the prediction accuracy and the improvement in utilization when applying the Bubble-Up methodology.

This is a challenging problem. The most relevant resource is a brute-force profiling approach, but with hundreds to thousands of applications running on the cluster, it is impractical.

We assume that pressure and sensitivity are the underlying causes of performance degradation due to arbitrary co-locations of production applications.

In this section, we describe how large-scale web-services co-locations in production datacenters.

The data includes the number of cores, amount of memory, application binary, associated data, and a configuration file for a task. Each task may also include special rules for configuration file.

To co-location, which is essential for co-location decision making.

We evaluate the prediction accuracy and the improvement in utilization when applying the Bubble-Up methodology.

Bubble-Up is a two-step characterization process. First, we measure the pressure on the memory subsystem. A metric reduces the complexity of co-location analysis. As this dial is increased, the pressure on the memory subsystem increases.

The bubble pressure score for the application using the loading suite, we also see from in Figure 2, that the sensitivity curve, A's pressure score is 2, we can predict the performance impact for a given application such as the one illustrated in Figure 2. On the y-axis, we have the normalized QoS performance of the host application, and on the x-axis, we have the number of applications being characterized. As this dial is increased, the precision of co-location leads to the heavy handed solution of simply disallowing co-location, which is essential for co-location decision making.

Jason Mars
(jasonmars.org)
The goal of this work is to enable the characterization methodology that enables the steering of pairwise co-locations of Google applications with at most a 2.2% error and often less than 1%. To evaluate using Bubble-Up to steer QoS performance degradation due to arbitrary co-locations of web-service and disk space that are to be allocated to the task. The configuration file for a task may also include special rules for enforcing co-locations of production workloads, we perform a signed stress test for the memory subsystem that provides a common pressure metric. Having such a metric reduces the complexity of co-location analysis. As an application suffers from different levels of pressure, the bubble's pressure score is a careful decomposition of the performance degradation that results from contention when co-located on production servers.

We study in a 500-machine cluster and are able to increase the aggressiveness they are for the shared memory resources and identify co-locations to reduce contention based on the classification of the performance degradation that results from co-location, which is essential for co-location decision prediction.

Bubble-Up is a two-step characterization process. First, when co-located, we evaluate the prediction accuracy and the improvement. To evaluate using Bubble-Up to steer QoS performance degradation due to arbitrary co-locations of production workloads does not violate this QoS threshold (light bars), we perform a signed stress test for the memory subsystem that provides a common pressure metric. Having such a metric reduces the complexity of co-location analysis. As an application suffers from different levels of pressure, the bubble's pressure score is a careful decomposition of the performance degradation that results from contention when co-located on production servers.

We study in a 500-machine cluster and are able to increase the aggressiveness they are for the shared memory resources and identify co-locations to reduce contention based on the classification of the performance degradation that results from co-location, which is essential for co-location decision prediction.
Bubble-Up!

In modern warehouse scale computers, each application is run in a datacenter, and the frequent development and updating of applications implies that a significant number of applications are run in a production datacenter environment. As applications are run on production servers, the co-locations of these applications may be contentious, both in terms of memory and in terms of the memory hierarchy used by the applications. When co-located with other applications, the performance degradation may be severe enough to disallow the co-location of the application in question, and instead force the application to be run on a separate machine. On the other hand, we could simply perform a stress test on an application, and if any of its co-locations result in performance degradation, simply run the application on the machine that maximizes the performance of the application.

The key insight of Bubble-Up is that predicting the performance impact for a given co-location is a challenging problem. The most relevant methodology is applied to each application, we have a sensitivity curve to look up the relative performance degradation that results from contention. The goal of this work is to enable the characterization methodology that enables the prediction of the performance impact for a given co-location. The characterization methodology is applied to each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.

We present the design of a methodology that enables the performance prediction of co-running applications. The methodology is a two-step characterization process. First, a sensitivity curve is associated with each application, and as this dial is increased automatically (expanding the bubble), the impact on the host system an application generates, and 2) measuring how much performance interference of co-running applications can be decoupled from utilization when applying the Bubble-Up methodology on the spectrum of applications.
Bubble-Up!

- Capture sensitivity and aggressiveness
  - Use bubble to produce sensitivity curve
  - Use reporter to produce aggressiveness score
The discretized sensitivity curve of application bubble sizes:

As we dial up, the bubble generates pressure on each shared resource. Because the pressure score is using the bubble’s pressure to co-location, which is essential for co-location decision. For the bubble, each design needs only to approximate varying levels of pressure, and there may be many good designs. This number is usually less than the number of cores available. The key insight of Bubble-Up is that predicting the performance degradation of an application in terms of the amount of pressure it causes on a shared cache, memory bandwidth or memory controller, can be allowed, resulting in improved utilization.

To evaluate using Bubble-Up to steer QoS, we introduce 17 production Google workloads and characterize every possible pairwise co-location, Bubble-Up only needs to approximate varying levels of pressure. The key to arrive at a good design that is not prone to error and underestimates pressure should result in worse performance.

We characterize methodology that enables the prediction of our Bubble-Up methodology on the spectrum of applications with hundreds to thousands of applications running. The frequent development and updating of our application being characterized. As this dial is increased automatically, different levels of pressure. The app’s degradation when running with an increasing amount of pressure on each shared resource.

To approximate varying levels of pressure, and there may be many good designs. This number is usually less than the number of cores available. The key insight of Bubble-Up is that predicting the performance degradation of an application in terms of the amount of pressure it causes on a shared cache, memory bandwidth or memory controller, can be allowed, resulting in improved utilization.

We introduce 17 production Google workloads and characterize every possible pairwise co-location, Bubble-Up only needs to approximate varying levels of pressure. The key to arrive at a good design that is not prone to error and underestimates pressure should result in worse performance.

We characterize methodology that enables the prediction of our Bubble-Up methodology on the spectrum of applications with hundreds to thousands of applications running. The frequent development and updating of our application being characterized. As this dial is increased automatically, different levels of pressure. The app’s degradation when running with an increasing amount of pressure on each shared resource.

We introduce 17 production Google workloads and characterize every possible pairwise co-location, Bubble-Up only needs to approximate varying levels of pressure. The key to arrive at a good design that is not prone to error and underestimates pressure should result in worse performance.

We characterize methodology that enables the prediction of our Bubble-Up methodology on the spectrum of applications with hundreds to thousands of applications running. The frequent development and updating of our application being characterized. As this dial is increased automatically, different levels of pressure. The app’s degradation when running with an increasing amount of pressure on each shared resource.
Bubble-Up!

- Stress test
- Provides a pressure dial
- Generate curve by increasing pressure
Bubble-Up!

- Stress test
- Provides a pressure dial
- Generate curve by increasing pressure

Step 1
- app
- app
- bubble
- bubble
- shared cache
- mem bandwidth
- DRAM

Step 2
- app
- app
- reporter
- core
- mem bandwidth
- shared cache
- DRAM

app’s sensitivity curve
output

app’s pressure score
output

app’s pressure

3.3 Step One: Characterizing Sensitivity

3.3.1 The Art of Bubble Design

- As the bubble’s pressure increases, the amount of interference should result in worse performance.
- Monotonic Curves

- Pressure dial

- Error stems from the mismatch of the relative sensitivities of each application due to the shared resources.
- The result is a set of pressure dials for the amount of pressure applied to the entire memory subsystem.

- Bubble-Up then uses the discretized sensitivity curve of an application to approximate an application’s pressure.
- Bubble-Up introduces a small amount of error when predicting the degradation in the memory subsystem using a pressure dial.
- To steer pairwise co-locations of Google applications, Bubble-Up is a two-step characterization process.

- In step 1 of Bubble-Up, we increase an application’s sensitivity to the pressure on the shared resource.
- In step 2, Bubble-Up automatically (expanding the bubble) the impact on the host machine utilization in the cluster by 50%–90%, depending on which cores the bubble is run.

- With this information, co-running application co-location will be more practical.

- The specific contributions of this work are as follows:
  - Using Bubble-Up, we are able to precisely predict the performance lost when co-located.
  - Bubble-Up only characterizes every possible pairwise co-location, providing a more precise prediction.
  - Bubble-Up only profits in the degradation of each application due to the shared resources.
  - Bubble-Up only generates on the shared resource for which this type of bubble design is applicable.

- Most importantly, Bubble-Up is a practical methodology to steer pairwise co-locations of Google applications.

- Figure 2: Example sensitivity curve for a “dial” for the amount of pressure applied to the entire memory subsystem.
**Bubble-Up!**

- Stress test
- Provides a pressure dial
- Generate curve by increasing pressure

**Step 1**
- app
- app
- bubble

**Step 2**
- app
- core

- reporter

**Description**

- **app's pressure dial**
- **shared cache**
- **mem bandwidth**
- **DRAM**

**Equation**

\[
\text{Deg}_{\text{app}}(B_i) = \sum_{j \in \{C, B, \ldots, B\}} \text{Deg}_{\text{app}}(B_j) \quad (2)
\]

- **app's pressure score**
- **output**

**Highly sensitive dummy**
- Observes the degradation of its own performance
- Reports *bubble* score
Bubble Design

\[
Deg_{AC} = \sum_{i}^{N} (S_{AR_i} \times P_{CR_i}) \\
\overline{Deg_{AC}} = Deg_{AB_K} \\
Error = \sum_{i}^{N} |S_{AR_i} \times P_{K_iR_i} - S_{AR_i} \times P_{CR_i}|
\]

A: app
C: co-runner
Bk: bubble at size k
Ri: shared resource i

- Select k, closest to C
- Error in proportions across Ri
- Hypothesis: Prioritize first order Rs and error will be small
- Good bubble design is key
Bubble Design

\[ Deg_{AC} = \sum_{i}^{N} (S_{AR_i} \times P_{CR_i}) \]

\[ \widehat{Deg}_{AC} = Deg_{ABK} \]

\[ Error = \sum_{i}^{N} |S_{AR_i} \times P_{BK_i} - S_{AR_i} \times P_{CR_i}| \]

A: app  
C: co-runner  
Bk: bubble at size k  
Ri: shared resource i

- Many possible designs
- Key Systematic Principles
  - “Monotonic Curves”
  - “Wide Dial Range”
  - “Broad Impact”

- Select k, closest to C
- Error in proportions across Ri
- Hypothesis: Prioritize first order  
  Rs and error will be small
- Good bubble design is key
Applying Bubble Up in Warehouse Scale Computers - An Illustrative Experiment

- Experimental Scenario
  - 500 machines running search-render, 6 core machines
  - 3 cores occupied by search render, 3 cores available for co-runner
  - 500 jobs ready to run, even mixture of the 17 Google workloads
  - Co-location decisions steered by Bubble-Up
Utilization Gained with Bubble-Up

Datacenter Utilization Improvement

Cluster Utilization

QoS Policy

better
Utilization Gained with Bubble-Up

Datacenter Utilization Improvement

Cluster Utilization

QoS Policy

 Improvement in cluster utilization when allowing Bubble-Up co-locations under each QoS policy.

Baseline 99% 98% 95% 90% 85% 80% max

Figure 200 shows the percentage of violations that cause less than 1% extra degradation beyond the QoS policy, meaning their QoS degradation is within a 98% QoS policy. However, all of these violations only cause less than 1% extra degradation beyond the QoS policy, around 10% of the co-locations violate the policy.

As Figure 200 demonstrates, Bubble-Up prediction greatly improves machine utilization. Even under 99% QoS policy, the utilization improvement is improved from 50% to close to 70%. Allowing a tolerance of just a few percent of violating co-locations when allowing all co-locations for each QoS policy reduces the number of violations. Note that most of the violations cause only less than 2% of extra QoS degradation. The QoS policy specifies the maximum tolerable QoS degradation when co-located applications are co-running on a machine. Then the co-location is close to 200. With 80% QoS policy, the co-location is close to 80%, showing great potential benefit of adopting Bubble-Up in datacenters.

The bubble-up's predication accuracy for pairwise co-locations of Google applications is close to 80%, showing great potential benefit of adopting Bubble-Up in datacenters.
Utilization Gained with Bubble-Up

Datacenter Utilization Improvement

Cluster Utilization

QoS Policy

- baseline
- 99%
- 98%
- 95%
- 90%
- 85%
- 80%
- max

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

better
Utilization Gained with Bubble-Up

- Uses Bubble-Up to steer co-location at various policies
Utilization Gained with Bubble-Up

- Uses Bubble-Up to steer co-location at various policies
- Significant improvements when allowing just 1% degradation
Utilization Gained with Bubble-Up

- Uses Bubble-Up to steer co-location at various policies
- Significant improvements when allowing just 1% degradation
Utilization Gained with Bubble-Up

- Uses Bubble-Up to steer co-location at various policies
- Significant improvements when allowing just 1% degradation
Utilization Gained with Bubble-Up

- Uses Bubble-Up to steer co-location at various policies
- Significant improvements when allowing just 1% degradation
- But what about violations?
Violations Eliminated with Bubble-Up

• Significant reduction in violations with bubble up.

• Violations are marginal

• QoS violated by only 1% - 3%
Violations Eliminated with Bubble-Up

- Significant reduction in violations with bubble up.
- Violations are marginal
- QoS violated by only 1% - 3%
Violations Eliminated with Bubble-Up

- Significant reduction in violations with bubble up.
- Violations are marginal
- QoS violated by only 1% - 3%

Percent of Violating Co-locations

![Graph showing percent of violating co-locations]

Increase in Co-locations

![Graph showing increase in co-locations]

QoS Enforced
QoS Violated <1%
QoS Violated 1%-2%
QoS Violated 2%-3%
QoS Violated >3%
QoS Enforced
Application to Other Platforms

- Same bubble used across all three platforms
- Curves and scores change, but same bubble/reporter design remains effective
- Spans 3 of the most widely used machine platforms in production
Application to Other Platforms

• Same bubble used across all three platforms

• Curves and scores change, but same bubble/reporter design remains effective

• Spans 3 of the most widely used machine platforms in production

More results in MICRO ’11 / TopPics ’12 Paper
Limitations of Bubble-Up
Limitations of Bubble-Up

- **Limitation 1** - Inability to adapt, which significantly limits utilization opportunities
Limitations of Bubble-Up

- **Limitation 1** - Inability to adapt, which significantly limits utilization opportunities
- **Limitation 2** - A priori knowledge required
Limitations of Bubble-Up

- **Limitation 1** - Inability to adapt, which significantly limits utilization opportunities
- **Limitation 2** - A priori knowledge required
- **Limitation 3** - Limited Co-location Scalability
Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers

Hailong Yang  Alex Breslow  Jason Mars  Lingjia Tang
University of California, San Diego
{h5yang, abreslow, mars, lingjia}@cs.ucsd.edu

ABSTRACT

Precisely providing quality of service for latency-sensitive applications while allowing co-locations in modern warehouse-scale computers (WSCs) is critical for improving utilization and reducing cost. Recent work relies on static profiling to precisely predict the performance interference among applications and the resulting QoS degradation for latency-sensitive applications. Based on the prediction, "safe" co-runners that do not violate QoS requirements can be identified to steer workload consolidation on each server, effectively improving server utilization while enforcing QoS requirements. However, these static profiling techniques have several critical limitations: 1) a priori knowledge of all workloads is required for profiling, 2) it is difficult for the prediction to capture or adapt to phase or load changes of applications, and 3) the prediction technique is limited to interference between only two applications.

To address these challenges, we present Bubble-PiPo: an integrated dynamic interference measurement and online QoS management mechanism to provide accurate QoS control and maximize server utilization. Bubble-PiPo uses a dynamic bubble to probe servers in real time and measure the instantaneous resource pressure to precisely predict how the QoS of the latency-sensitive job will be affected by potential co-runners to instantaneously identify safe co-runners. Once batch jobs are selected and mapped to a server, Bubble-PiPo uses an online phase-in/phase-out (PiPo) mechanism to continuously monitor the QoS of the latency-sensitive application and adaptively control the execution of batch jobs to adapt to dynamic input, phase, and load changes to deliver satisfactory QoS. The dynamic bubble and online PiPo are able to manage multiple applications without the requirement of a priori knowledge. Bubble-PiPo provides a holistic, scalable and adaptive solution for maximizing utilization in WSCs.

1. INTRODUCTION

Improving utilization in modern warehouse-scale computers (WSCs) has been identified as a critical design goal for reducing the total cost of ownership for webserver providers [1]. However, the over-provisioning of compute resources in WSCs to ensure high quality-of-service (QoS) of latency sensitive applications such as websearch continues to be prohibitive in realizing high utilization.

Arguably the most challenging obstacle to improving utilization in modern WSCs is in providing precise QoS prediction and management for latency-sensitive applications when co-running with other applications. The lack of software and/or hardware mechanisms to dynamically and instantaneously detect precisely how jobs interfere when co-located on a single server has resulted in WSC operators simply disallowing co-location for user-facing latency sensitive applications. This challenge is a major contributor to utilization in WSCs and suggests why average utilization is under 50% even at large companies such as Microsoft and Google [3], and a recent article from Wired reported that Mozilla’s data centers operate at 6% utilization, and VMWare’s at 20 to 30% [33].

Recent work [26, 33] has proposed a static profiling technique to precisely predict the QoS interference and degradation for latency-sensitive applications when co-located. Based on the prediction, the cluster scheduler can identify batch applications that provide “safe colocations”, effectively improving server utilization while enforcing QoS requirements of latency-sensitive applications. However, this technique has several limitations that critically affect its generality and effectiveness. These include:

1. A static technique requires a priori knowledge about all workloads and requires profiling for each type of workload, which limits both the types of workloads for which such a technique can be applied and, more broadly, the type of datacenters that can adopt the approach.

2. A static profiling and prediction technique cannot capture and enable adaptation to an application’s phase, input and load as they change during execution and across executions.

3. The techniques provided in prior work are limited to predicting interference between two co-located applications. It remains unclear how such techniques can scale up to predict the QoS degradation caused by three or more applications.
Machine Level

- Gen 1 Xeon: 0 Jobs
- Gen 2 Xeon: 1 Job Running
- Opteron: 3 Jobs Running
Machine Level
Application Rigidness at the Machine Level

Compiler

<table>
<thead>
<tr>
<th>P4</th>
<th>Core i7</th>
<th>Core i7</th>
<th>Core i7</th>
<th>Core i7</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE 1</td>
<td>Athlon</td>
<td>Athlon</td>
<td>Athlon</td>
<td>Athlon</td>
</tr>
</tbody>
</table>

EE 2

EE 3
Application Rigidness at the Machine Level

Compiler

Web Search

P4

EE 1

Core i7 | Core i7 | Core i7 | Core i7

EE 2

Athlon | Athlon | Athlon | Athlon

EE 3

Co-Runner | Co-Runner | Co-Runner

Tuesday, October 8, 13
Application Rigidness at the Machine Level

Compiler

- EE 1
  - P4
  - Core i7
- EE 2
  - Core i7
  - Core i7
- EE 3
  - Core i7
  - Athlon
  - Co-Runner
  - Co-Runner
  - Co-Runner
  - Co-Runner

Web Search
Application Rigidness at the Machine Level

Compiler

P4

Core i7  Core i7  Core i7  Core i7

EE 1

EE 2

EE 3

Co-Runner  Co-Runner  Co-Runner

Athlon  Athlon  Athlon  Athlon
Application Rigidness at the Machine Level

- Problem: Aggressive Compiler Optimizations
Application Rigidness at the Machine Level

- Problem: Aggressive Compiler Optimizations
Application Rigidness at the Machine Level

- Problem: Aggressive Compiler Optimizations

- Compiler

- Web Search

- P4

- Core i7  Core i7  Core i7  Core i7

- Co-Runner  Co-Runner  Co-Runner

- Athlon  Athlon  Athlon  Athlon

- EE 1

- EE 2

- EE 3
Application Rigidness at the Machine Level

- Problem: Aggressive Compiler Optimizations

Compiler

Web Search

P4

Core i7 Core i7 Core i7 Core i7

EE 1 EE 2 EE 3

Co-Runner Co-Runner Co-Runner

Athlon Athlon Athlon Athlon

Job Mapper

Gen 1 Xeon 0 Jobs

Gen 2 Xeon 1 Job Running

Opteron 3 Jobs Running

• Problem: Aggressive Compiler Optimizations

Tuesday, October 8, 13
Application Rigidness at the Machine Level

- Problem: Aggressive Compiler Optimizations

- Compiler
  - Web Search
  - P4: Core i7, Core i7, Core i7, Core i7
  - EE 1: Core i7, Core i7, Core i7, Core i7
  - EE 2: Athlon, Athlon, Athlon, Athlon
  - EE 3: Athlon, Athlon, Athlon, Athlon

Tuesday, October 8, 13
Application Rigidness at the Machine Level

- Problem: Aggressive Compiler Optimizations

```
  Compiler

  P4       Core i7  Core i7  Core i7  Core i7
  EE 1     EE 2     EE 3

  Web Search  Co-Runner  Co-Runner  Co-Runner
  Athlon      Athlon      Athlon      Athlon
```

Tuesday, October 8, 13
Application Rigidness at the Machine Level

- Problem: Aggressive Compiler Optimizations

Compiler

<table>
<thead>
<tr>
<th>Web Search</th>
<th>Co-Runner</th>
<th>Co-Runner</th>
<th>Co-Runner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Athlon</td>
<td>Athlon</td>
<td>Athlon</td>
<td>Athlon</td>
</tr>
</tbody>
</table>

P4

EE 1

Core i7 Core i7 Core i7 Core i7

EE 2

Core i7 Core i7 Core i7 Core i7

EE 3

Core i7 Core i7 Core i7 Core i7
Application Rigidness at the Machine Level

- Problem: Aggressive Compiler Optimizations
- Requirement: Mechanism for cracking the rigidness
Realizing Online Adaptation
Realizing Online Adaptation

• Traditional Approach: Dynamic binary translators
  • Huge complexity, sacrifice performance
Realizing Online Adaptation

- Traditional Approach: Dynamic binary translators
  - Huge complexity, sacrifice performance
- To achieve deployability within WSCs
  - Must be lightweight
  - Low complexity
Realizing Online Adaptation

• Traditional Approach: Dynamic binary translators
  • Huge complexity, sacrifice performance
• To achieve deployability within WSCs
  • Must be lightweight
  • Low complexity
• We need new technology for online adaptation
Realizing Online Adaptation

• Traditional Approach: Dynamic binary translators
  • Huge complexity, sacrifice performance

• To achieve deployability within WSCs
  • Must be lightweight
  • Low complexity

• We need new technology for online adaptation

• This new technology is one of the major challenges and contributions of this work
New Approach to Adaptation

Conventional Binary Adaptation
New Approach to Adaptation

Conventional Binary Adaptation

Application
New Approach to Adaptation

Conventional Binary Adaptation

Application

Runtime

Tuesday, October 8, 13
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level

Application

Code Cache

Runtime

Tuesday, October 8, 13
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
- High complexity, High overhead
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
- High complexity, High overhead

New Approach for Binary Adaptation
New Approach to Adaptation

**Conventional Binary Adaptation**

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
- High complexity, High overhead

**New Approach for Binary Adaptation**
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
- High complexity, High overhead

New Approach for Binary Adaptation

Application
Code Cache
Runtime
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
- High complexity, High overhead

New Approach for Binary Adaptation

- No fine grain control, execute directly
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
- High complexity, High overhead

New Approach for Binary Adaptation

- No fine grain control, execute directly
- Statically stitch in adaptability
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
- High complexity, High overhead

New Approach for Binary Adaptation

- No fine grain control, execute directly
- Statically stitch in adaptability
New Approach to Adaptation

**Conventional Binary Adaptation**
- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
- High complexity, High overhead

**New Approach for Binary Adaptation**
- No fine grain control, execute directly
- Statically stitch in adaptability
- Enable coordination between runtimes
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
- High complexity, High overhead

New Approach for Binary Adaptation

- No fine grain control, execute directly
- Statically stitch in adaptability
- Enable coordination between runtimes
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
- High complexity, High overhead

New Approach for Binary Adaptation

- No fine grain control, execute directly
- Statically stitch in adaptability
- Enable coordination between runtimes
- Low complexity, Low overhead
New Approach to Adaptation

Conventional Binary Adaptation

- Fine grain control at instruction/BB level
- Expensive transitions from runtime to code
- Instrumentation for monitoring of EE
- High complexity, High overhead

New Approach for Binary Adaptation

- No fine grain control, execute directly
- Statically stitch in adaptability
- Enable coordination between runtimes
- Low complexity, Low overhead

Loaf: Lightweight Online Adaptation Framework
Loaf: Adapting Code

• Requires Loaf SBM enabled Compiler (Integrated in GCC 4.3)

• Generated binary hooks into introspection engine
Loaf: Adapting Code

- Requires Loaf SBM enabled Compiler (Integrated in GCC 4.3)
- Generated binary hooks into introspection engine
Loaf: Adapting Code

- Requires Loaf SBM enabled Compiler (Integrated in GCC 4.3)

- Generated binary hooks into introspection engine
Loaf: Adapting Code

• Requires Loaf SBM enabled Compiler (Integrated in GCC 4.3)

• Generated binary hooks into introspection engine
Loaf: Monitoring
Loaf: Monitoring

- Thin runtime layer
Loaf: Monitoring

- Thin runtime layer

- No instrumentation for monitoring, Only use performance monitoring unit
Loaf: Monitoring

• Thin runtime layer

• No instrumentation for monitoring, Only use performance monitoring unit

• Uses a periodic probing mechanism
Loaf: Monitoring

• Thin runtime layer

• No instrumentation for monitoring, Only use performance monitoring unit

• Uses a periodic probing mechanism

• Probes at ~1ms (very low overhead)
SBO Adaptation Policy

- Problem: Aggressive optimizations may improve or degrade performance

- Attack: Use Loaf to enact competition heuristic

  - Learn whether aggressive is beneficial
SBO Adaptation Policy

• Problem: Aggressive optimizations may improve or degrade performance

• Attack: Use Loaf to enact competition heuristic

• Learn whether aggressive is beneficial
SBO Adaptation Policy

• Problem: Aggressive optimizations may improve or degrade performance

• Attack: Use Loaf to enact competition heuristic

  • Learn whether aggressive is beneficial
SBO Adaptation Policy

• Problem: Aggressive optimizations may improve or degrade performance

• Attack: Use Loaf to enact competition heuristic

  • Learn whether aggressive is beneficial
SBO Adaptation Policy

- Problem: Aggressive optimizations may improve or degrade performance
- Attack: Use Loaf to enact competition heuristic
  - Learn whether aggressive is beneficial

<table>
<thead>
<tr>
<th>Learning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>phase 1: $T_1$</td>
<td>phase 2: $T_1$</td>
</tr>
</tbody>
</table>
SBO: Results

Impact of OAAAO on Execution Time

80% 90% 100% 110% 120% 130%

Execution Time (normalized)

better
SBO: Results

Impact of OAAAO on Execution Time

Execution Time (normalized)

80% 90% 100% 110% 120% 130%

better

433.milc 444.namd 453.povray 470.lbm 401.bzip2 403.gcc 429.mcf 445.gobmk 456.hmmer 462.libquantum 464.h264ref 473.astar mean

O2 O2+pref O2+unroll O2+pref+unroll O2+oaaao

Job Mapper
Gen 1 Xeon 0 Jobs
Gen 2 Xeon 1 Job Running
Opteron 3 Jobs Running

Tuesday, October 8, 13
SBO: Results

Impact of OAAAO on Execution Time

Performance 4-10% better
SBO: Results

Impact of OAAAO on Execution Time

- Execution Time (normalized)
- More results in CGO ‘09 Paper

Performance: 4-10%

- O2
- O2+pref
- O2+unroll
- O2+pref+unroll
- O2+oaaao

Better performance:

Tuesday, October 8, 13
In Production GCC 4.8

Function Multiversioning

Created November 12, 2012

Description

This support has been checked in to trunk and should be available when GCC 4.8 is released. Support is only available in C++ for x86 targets.

Frequently executed functions in applications are sometimes built into many versions to take advantage of specific support or features of the hardware that executes the application. For example, functions are compiled to use SSE4 instructions if the hardware supports it. There is, however, the developer burden of creating the dispatching mechanism to execute the right version at runtime. This aim of this project is to make it really easy for the developer to specify multiple versions of a function, each catering to a specific target ISA feature. GCC then takes care of creating the dispatching code necessary to execute the right function version. With this support, here is a simple example of how to create function versions:

```c
_attribute__ ((target ("default")))
int foo ()
{
  // The default version of foo.
  return 0;
}

_attribute__ ((target ("sse4.2")))
int foo ()
{
  // foo version for SSE4.2
  return 1;
}

_attribute__ ((target ("arch=atom")))
int foo ()
{
```
In Production GCC 4.8

Function Multiversioning

Description

This support has been checked in to trunk and should be available when GCC 4.8 is released. Support is only available in C++ for i386 targets.

Frequently executed functions in applications are some times built into many versions to take advantage of specific support or features of the hardware that executes the application. For example, functions are compiled to use SSE4 instructions if the hardware supports it. There is, however, the developer burden of creating the dispatching mechanism to execute the right version at runtime. This aim of this project is to make it really easy for the developer to specify multiple versions of a function, each catered to a specific target ISA feature. GCC then takes care of creating the dispatching code necessary to execute the right function version. With this support, here is a simple example of how to create function versions:

```
__attribute__((target("default")))
int foo ()
{
  // The default version of foo.
  return 0;
}
__attribute__((target("sse4.2")))
int foo ()
{
  // foo version for SSE4.2
  return 1;
}
__attribute__((target("arch=atom")))
int foo ()
{

```

the versions should be dispatched? This is answered by

the version targeted for SSE2, that is, function versions with

the target features is determined by the target.

None: FunctionMultiversioning (last edited 2013-02-04 22:31:54 by SriramTallam)
In Production GCC 4.8

Function Multiversioning

This support has been checked into trunk and should be available when GCC 4.8 is released. Support is only available in C++ for i386 targets.

Frequently executed functions in applications are sometimes built into many versions to take advantage of specific support or features of the hardware that executes the application. For example, functions are compiled to use SSE4 instructions if the hardware supports it. There is, however, the developer burden of creating the dispatching mechanism to execute the right version at runtime. This aim of this project is to make it really easy for the developer to specify multiple versions of a function, each catering to a specific target ISA feature. GCC then takes care of creating the dispatching code necessary to execute the right function version. With this support, here is a simple example of how to create function versions.

```c
_attribute__((target("default")))
int foo ()
{
// The default version of foo.
   return 0;
}

_attribute__((target("sse4.2")))
int foo ()
{
// foo version for SSE4.2
   return 1;
}

_attribute__((target("arch=atom")))
int foo ()
{
   // foo version for Atom
   return 2;
}
```

The versions should be dispatched? This is answered by relying on the target attributes. For example, a function version in a version targeted for SSE2, that is, function versions with the target features is determined by the target.

None: FunctionMultiversioning (last edited 2013-02-04 22:31:54 by SriramanTallam)
Machine Level Adaptation Insights

• Embrace vertical integration (compiler + runtime + architecture)
Machine Level Adaptation Insights

• Embrace vertical integration (compiler + runtime + architecture)

• Involve compiler to “stitch in” reconfigurability
Machine Level Adaptation Insights

• Embrace vertical integration (compiler + runtime + architecture)

• Involve compiler to “stitch in” reconfigurability

• Performance monitors is a key enabler of online software techniques
Machine Level Adaptation Insights

• Embrace vertical integration (compiler + runtime + architecture)

• Involve compiler to “stitch in” reconfigurability

• Performance monitors is a key enabler of online software techniques

• Approaches presented are realizable today, and will gain more traction as performance counters integrated in ABI
Machine Level Adaptation Insights

• Embrace vertical integration (compiler + runtime + architecture)

• Involve compiler to “stitch in” reconfigurability

• Performance monitors is a key enabler of online software techniques

• Approaches presented are realizable today, and will gain more traction as performance counters integrated in ABI

• Day will come where all code and systems will continually be restructuring - like warming of a cache
Cluster Level

Job Mapper

Gen 1 Xeon
0 Jobs

Gen 2 Xeon
1 Job Running

Opteron
3 Jobs Running
Cluster Level
The Folly of the Homogenous Assumption
The Folly of the Homogenous Assumption

Figure 4.1: The amount of platform diversity found in 10 randomly selected anonymized Google datacenters in operation. As shown in the figure, these 10 datacenters house as few as two and as many as five different microarchitectural configurations, including both Intel and AMD servers from several consecutive generations. Yet, the assumption of homogeneity has been a core design philosophy behind the job management subsystems of modern WSCs. As Figure 4.2 shows, the job manager views the WSC as a collection of tens to hundreds of thousands of cores with the assumption of homogeneity. Available machine resources are assigned to jobs according to their core and memory requirement. The diversity across the underlying microarchitectures in the WSC is not explicitly considered by the job management subsystem. However, as we show in this work, ignoring this heterogeneity leads to inefficient execution of applications in these datacenters.
The Folly of the Homogenous Assumption

Homogenous Cores Assumption (Job Managers View)

Actual Machines are Heterogeneous

Figure 3.1: Chapter 3. Execution Milieus in WSCs

Figure 3.2: Chapter 3. Execution Milieus in WSCs

Figure 4.2: Chapter 4. Mapping Jobs to Exploit Diversity in Execution Milieus
Exploiting this E.E. Diversity
Exploiting this E.E. Diversity

• Goal

  • Incorporate awareness of this EE diversity.

  • Map jobs accordingly
Exploiting this E.E. Diversity

• Goal

  • Incorporate awareness of this EE diversity.

  • Map jobs accordingly

• Leverage unique properties of WSCs

  • Set of known applications running continuously

  • Continuous profiling service (GWP)
Continuous Profiling in WSCs
Continuous Profiling in WSCs

• Google Wide Profiling
  • Functions as a distributed service
  • Samples all binaries running across the fleet
  • Primarily uses hardware performance counters
  • Very lightweight (less than 0.01% overhead) and always on
  • Used by humans for debugging and performance analysis
Continuous Profiling in WSCs

- Google Wide Profiling
  - Functions as a distributed service
  - Samples all binaries running across the fleet
  - Primarily uses hardware performance counters
  - Very lightweight (less than 0.01% overhead) and always on
  - Used by humans for debugging and performance analysis
- Tip of the iceberg of such tools
Continuous Profiling in WSCs

• Google Wide Profiling
  • Functions as a distributed service
  • Samples all binaries running across the fleet
  • Primarily uses hardware performance counters
  • Very lightweight (less than 0.01% overhead) and always on
  • Used by humans for debugging and performance analysis

• Tip of the iceberg of such tools

• Key Insight
  • Leverage these tools for automated learning and action
  • Indeed, it’s big data
Whare-Map: **WSC Heterogeneity Aware Mapper**
Whare-Map: WSC Heterogeneity Aware Mapper

Figure 4: Performance comparison of benchmark workloads across three microarchitectures.

Figure 5: Benchmark slowdown when co-located applications present various levels of performance variability of an application mix and the machine mix are known.

Figure 6: The Overview of Whare-Map

Job Manager

Whare-Map

Optimization Solver

Map Scorer

Application

Machine

Application

Machine

... Application

Machine

Datacenter

GWP
Whare-Map: **WSC Heterogeneity Aware Mapper**

- Whare-Map
  - Build knowledge bank of Job types and relative performance in various EE
  - Continuously train map scorer with GWP
Whare-Map: **WSC Heterogeneity Aware Mapper**

- Whare-Map
  - Build knowledge bank of Job types and relative performance in various EE
- Continuously train *map scorer* with GWP

- Google Wide Profiler (GWP)
  - Continuously samples all machines
  - Collects hardware performance counter information
Formulating Mapping as an Optimization Problem
Formulating Mapping as an Optimization Problem

- We formulate mapping as an optimization problem
Formulating Mapping as an Optimization Problem

- We formulate mapping as an optimization problem

- Construct a model based on information collected in our knowledge bank
  
  - Each job placement has a score associated with it
  
  - Aggregate all scores for overall map score
Formulating Mapping as an Optimization Problem

- We formulate mapping as an optimization problem
- Construct a model based on information collected in our knowledge bank
- Each job placement has a score associated with it
- Aggregate all scores for overall map score
- We use a simple stochastic hill climbing, works well

**Algorithm 1: Core Optimization Algorithm**

```
Input: set of free machines and available jobs
Output: an optimized mapping

1. while free machines and available jobs do
2.     map random job to random machine;
3. end
4. set last_score to the score of current map;
5. while optimization timer not exceeded do
6.     foreach machine do
7.         foreach job on that machine do
8.             swap job with random job on random machine;
9.             set cur_score to the score of current map;
10.            if mapping score is better then
11.                set last_score to cur_score;
12.            else
13.                swap jobs back to original placements;
14.            end
15.         end
16.     end
17. end
```
# Building the Knowledge Bank

![Image](image.png)

## Table 5: Mapping Scoring Policies

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whare-C</td>
<td><strong>Colocation Score</strong>: This score is based only on co-location penalty and only requires profiling the co-location penalty on any type of machine. Once a co-location profile is collected it is then used to score that co-location regardless of the underlying microarchitecture.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-Cs</td>
<td><strong>Colocation Score (Smart)</strong>: This score is based on co-location penalty with microarchitecture specific information. Information about co-location penalty must be collected for all platforms of interest.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-M</td>
<td><strong>Microarchitectural Affinity Score</strong>: This score is based on microarchitectural affinity and captures only the speedup of running each application on one microarchitecture over another.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-MCs</td>
<td><strong>Microarchitectural Affinity and Colocation Score</strong>: This scoring method includes both microarchitectural affinity and microarchitecture specific co-location penalty. This scoring technique has the heaviest profiling requirements.</td>
<td>$</td>
</tr>
</tbody>
</table>
Building the Knowledge Bank

- As time passes, quality of knowledge bank improves

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whare-C</td>
<td><strong>Colocation Score</strong>: This score is based only on co-location penalty and only requires profiling the co-location penalty on any type of machine. Once a co-location profile is collected it is then used to score that co-location regardless of the underlying microarchitecture.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-Cs</td>
<td><strong>Colocation Score (Smart)</strong>: This score is based on co-location penalty with microarchitecture specific information. Information about co-location penalty must be collected for all platforms of interest.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-M</td>
<td><strong>Microarchitectural Affinity Score</strong>: This score is based on microarchitectural affinity and captures only the speedup of running each application on one microarchitecture over another.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-MCs</td>
<td><strong>Microarchitectural Affinity and Colocation Score</strong>: This scoring method includes both microarchitectural affinity and microarchitecture specific co-location penalty. This scoring technique has the heaviest profiling requirements.</td>
<td>$</td>
</tr>
</tbody>
</table>
Building the Knowledge Bank

- As time passes, quality of knowledge bank improves

- We look at 4 classes of knowledge bank (scoring policies)
  - Whare-C
  - Whare-Cs
  - Whare-M
  - Whare-MCs

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whare-C</td>
<td><strong>Colocation Score</strong>: This score is based only on co-location penalty and only requires profiling the co-location penalty on any type of machine. Once a co-location profile is collected it is then used to score that co-location regardless of the underlying microarchitecture.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-Cs</td>
<td><strong>Colocation Score (Smart)</strong>: This score is based on co-location penalty with microarchitecture specific information. Information about co-location penalty must be collected for all platforms of interest.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-M</td>
<td><strong>Microarchitectural Affinity Score</strong>: This score is based on microarchitectural affinity and captures only the speedup of running each application on one microarchitecture over another.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-MCs</td>
<td><strong>Microarchitectural Affinity and Colocation Score</strong>: This scoring method includes both microarchitectural affinity and microarchitecture specific co-location penalty. This scoring technique has the heaviest profiling requirements.</td>
<td>$</td>
</tr>
</tbody>
</table>

Where-MCs
Building the Knowledge Bank

- As time passes, quality of knowledge bank improves
- We look at 4 classes of knowledge bank (scoring policies)
  - Whare-C
  - Whare-Cs
  - Whare-M
  - Whare-MCs

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whare-C</td>
<td><strong>Colocation Score:</strong> This score is based only on co-location penalty and only requires profiling the co-location penalty on any type of machine. Once a co-location profile is collected it is then used to score that co-location regardless of the underlying microarchitecture.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-Cs</td>
<td><strong>Colocation Score (Smart):</strong> This score is based on co-location penalty with microarchitecture specific information. Information about co-location penalty must be collected for all platforms of interest.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-M</td>
<td><strong>Microarchitectural Affinity Score:</strong> This score is based on microarchitectural affinity and captures only the speedup of running each application on one microarchitecture over another.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-MCs</td>
<td><strong>Microarchitectural Affinity and Colocation Score:</strong> This scoring method includes both microarchitectural affinity and microarchitecture specific co-location penalty. This scoring technique has the heaviest profiling requirements.</td>
<td>$</td>
</tr>
</tbody>
</table>
Building the Knowledge Bank

- As time passes, quality of knowledge bank improves
- We look at 4 classes of knowledge bank (scoring policies)
  - Whare-C
  - Whare-Cs
  - Whare-M
  - Whare-MCs

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whare-C</td>
<td><strong>Colocation Score</strong>: This score is based only on co-location penalty and only requires profiling the co-location penalty on any type of machine. Once a co-location profile is collected it is then used to score that co-location regardless of the underlying microarchitecture.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-Cs</td>
<td><strong>Colocation Score (Smart)</strong>: This score is based on co-location penalty with microarchitecture specific information. Information about co-location penalty must be collected for all platforms of interest.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-M</td>
<td><strong>Microarchitectural Affinity Score</strong>: This score is based on microarchitectural affinity and captures only the speedup of running each application on one microarchitecture over another.</td>
<td>$</td>
</tr>
<tr>
<td>Whare-MCs</td>
<td><strong>Microarchitectural Affinity and Colocation Score</strong>: This scoring method includes both microarchitectural affinity and microarchitecture specific co-location penalty. This scoring technique has the heaviest profiling requirements.</td>
<td>$</td>
</tr>
</tbody>
</table>
Experimental Setup

- 500 machines
- 3 production machine configuration types
- 1000 jobs
- 9 Google applications (job types)
- Perflab benchmark suite
- Experimental testbed using SPEC2006

<table>
<thead>
<tr>
<th>CPU</th>
<th>GHz</th>
<th>Cores</th>
<th>L2/L3</th>
<th>Nickname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clovertown Xeon E5345</td>
<td>2.33GHz</td>
<td>6</td>
<td>8mb</td>
<td>Clover</td>
</tr>
<tr>
<td>Istanbul Opteron 8431</td>
<td>2.4GHz</td>
<td>6</td>
<td>6mb</td>
<td>Istan</td>
</tr>
<tr>
<td>Westmere Xeon X5660</td>
<td>2.8GHz</td>
<td>6</td>
<td>12mb</td>
<td>West</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>workload</th>
<th>description</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigtable</td>
<td>A distributed storage system for managing petabytes of structured data</td>
<td>user-facing</td>
</tr>
<tr>
<td>ads-servlet</td>
<td>Ads sever responsible for selecting and placing targeted ads on syndication partners sites</td>
<td>user-facing</td>
</tr>
<tr>
<td>maps-detect-face</td>
<td>Face detection for streetview automatic face blurring</td>
<td>batch</td>
</tr>
<tr>
<td>search-render</td>
<td>Websearch frontend server, collect results from many backends and assembles html for user.</td>
<td>user-facing</td>
</tr>
<tr>
<td>search-scoring</td>
<td>Websearch scoring and retrieval</td>
<td>user-facing</td>
</tr>
<tr>
<td>protobuf</td>
<td>Protocol Buffer, a mechanism for describing extensible communication protocols and on-disk structures. One of the most commonly-used programming abstractions at Google.</td>
<td>user-facing</td>
</tr>
<tr>
<td>docs-analyzer</td>
<td>Unsupervised Bayesian clustering tool to take keywords or text documents and “explain” them with meaningful clusters.</td>
<td>both</td>
</tr>
<tr>
<td>saw-countw</td>
<td>Sawzall scripting language interpreter benchmark</td>
<td>both</td>
</tr>
<tr>
<td>youtube-x264yt</td>
<td>x264yt video encoding.</td>
<td>batch</td>
</tr>
</tbody>
</table>
Results: Overall IPS improvement

- IPS - instructions per second
- Improvement - Google: 16%, SPEC: 14%
Results: Overall IPS improvement

- IPS - instructions per second
- Improvement - Google: 16%, SPEC: 14%
Results: Overall IPS improvement

- IPS - instructions per second
- Improvement - Google: 16%, SPEC: 14%
What Kind of Machines Should I Buy?

- When applying SmartyMap...
  - It may be much more cost effective to buy a mixture of machines
  - Notice Cover vs Cover+Istan, and West vs Istan+West
What Kind of Machines Should I Buy?

- When applying SmartyMap...
  - It may be much more cost effective to buy a mixture of machines
  - Notice Cover vs Cover+Istan, and West vs Istan+West
What Kind of Machines Should I Buy?

- When applying SmartyMap...
  - It may be much more cost effective to buy a mixture of machines
  - Notice Cover vs Cover+Istan, and West vs Istan+West

More results in ISCA ’13 Paper
In Production

![Graph showing speedup (IPS) for various applications]

- **D0**: Significant improvement of up to 15%
- **D1**: Achieves little over 10% improvement
- **D2** to **D9**: Various levels of improvement, with **D5** showing the highest improvement

### 5.4 Factors Impacting Heterogeneity in WSCs

1. **Performance Opportunity Calculation**: Based on this paper, we now report measured significant potential improvement of up to 15% when intelligently placing the remaining jobs. This performance opportunity calculation is based on the assumption of heterogeneity available in the WSC along two dimensions.

2. **Whare-Map's Performance Improvement**: Whare-Map's performance improvement over the heterogeneity-oblivious mapping is approximately 10%.

3. **Factors Affecting Heterogeneity**: Heterogeneity varies when the workload mix varies. Specifically, we have the following observations and insights.

   - **CPU Intensive Workloads**: Most advantages of heterogeneity are observed in CPU intensive workloads. In Figure 9, both 1J-CPU and 2J-CPU experiments have relatively low performance improvement. This indicates that for CPU intensive workloads, the advantage of heterogeneity varies when the workload mix varies. In Figure 10, the opportunity factor of each application varies from a lack of understanding on how the gradual introduction of a month of operation impacts performance variability.

   - **Memory Intensive Workloads**: The benchmark testbed experiments use 400 machines composed of 3 types of microarchitectures listed in Table 3. Similarly, the frequency benefit potential is smaller for CPU intensive benchmarks. The y-axis shows the calculated opportunity factor of each application. Figure 11 shows the performance opportunity from the heterogeneity-oblivious mapping, as opposed to close to 15% for memory intensive benchmarks. The x-axis shows the opportunity factor of each application.

### Job Mapper

- **Job Running**: 1 Job Running
- **Jobs**: 0 Jobs
- **Job Mapper**
Whare-Map Wrap-up

- *Emergent* heterogeneity should not be ignored

- Continuous monitoring services (e.g., GWP) not only for debugging

- ~15% improvement with minimal worst case placements

- More in the Paper!

  - New metric: Opportunity Factor

  - Extensive experimentation on both Google and Benchmark workloads

  - Analysis of map scoring policies
Key Take-Aways
Key Take-Aways

Bridging the software/hardware gap

Understand and exploit the interaction between emerging WSC workload and the underlying HW
Key Take-Aways

Bridging the software/hardware gap

Understand and exploit the interaction between emerging WSC workload and the underlying HW

Change the way we think about Optimization

quality of service
co-running applications
server utilization
power/energy

performance only
Questions
Impact

• Almost 3000 downloads of my work (ACM Digital Library)
  • > 1000 downloads just this year

• SBO picked up by Google, full time engineers working on it, in production GCC 4.8

• CAER feature on Google Research Blog

• Bubble-Up recognized for “novelty and long-term impact” (TopPicks 2012)

• BlockChop used in ongoing development of Hybrid chip at Intel

• ISCA ’11 paper featured by Google as one of the years best
Backup Slides
Expensive? How?

• Where’s the money going?
Expensive? How?

• Where’s the money going?

Figure 6.1 shows a breakdown of the yearly TCO for case A among datacenter and server-related Opex and Capex components. In this example, which is typical of classical datacenters, the high server capital costs dominate overall TCO, with 69% of the monthly cost related to server purchase and maintenance. However, commodity-based lower-cost (and perhaps lower-reliability) servers, or higher power prices, can change the picture quite dramatically. For case B (see Figure 6.2), we assume a cheaper, faster, higher-powered server consuming 500 W at peak and costing only $2,000 in a location where electricity cost is $0.10/kWh. In this case, datacenter-related costs rise to 49% of the total, and energy costs to 22%, with server costs falling to 29%. In other words, the hosting cost of such a server, that is, the cost of all infrastructure and power to house it, is more than twice the cost of purchasing and maintaining the server in this scenario.

Note that even with the assumed higher power price and higher server power, the absolute 3-year TCO in case B is lower than in case A ($8,702 vs $10,757) because the server is so much cheaper. The relative importance of power-related costs is likely to increase as shown in case B.

An online version of the spreadsheet underlying the graphs in this section is at http://spreadsheets.google.com/pub?key=phRJ4tNx2bFOHgYskgpoXAA&output=xls.

Figure: TCO (Total Cost of Ownership) cost breakdown for a datacenter using commodity servers *

* Barroso et al, “The datacenter as a computer: An introduction to the design of warehouse-scale machines”, Synthesis Lectures on Computer Architecture ’09
Lessons Learned

- Allow some cross layer awareness (compiler + runtime + architecture)

- Involve compiler to “stitch in” reconfigurability

- Performance monitors are the future of online software techniques

- Approaches presented are a little ahead of time because of state of hardware performance counters

- Day will come where all code and systems will continually be restructuring - like warming of a cache
Bubble Design: The Sensitivity of Google Applications

![Sensitivity Curves for different Google applications](jasonmars.org)
Prediction Accuracy of Bubble-Up

bigtable, avg. error - 2.2%
ads-servlet, avg. error - 0.8%
maps-detect-face, avg. error - 0.7%
search-renderer, avg. error - 1.8%
search-scoring, avg. error - 0.8%
protobuf, avg. error - 2.2%
docs-analyzer, avg. error - 1.7%
saw-count2, avg. error - 1.2%
youtube-x264yt, avg. error - 1.5%
Google Apps Co-located with Google Apps

Bubble-Up’s predication accuracy for pairwise co-locations of Google applications.

- Prediction error when co-locating Google applications
- Errors in positive direction do not lead to violations
Google Apps Co-located with Google Apps

Bubble-Up’s prediction accuracy for pairwise co-locations of Google applications.

- Prediction error when co-locating Google applications
- Errors in positive direction do not lead to violations
Experimental Setup

- Workloads
  - Suite of 17 Google Workloads
    - Spans search, maps, docs, etc
    - Replays log of production Queries, representative of peak load
  - SmashBench
    - Suite of contentious kernels with a spectrum of properties
    - Developed at Google specifically for studying contention

- Platforms
  - Production machines in Google’s fleet
  - 6-core server-grade Xeons and Opterons
Loaf: Adapting Environment

- Environment composed of co-running applications
- Coordinate adaptation using shared communication table
- Coordination feature built into Introspection Engine
CAER: Contention Lead to Performance Interference

- Current solution: simply disallow co-location

- Sacrifices utilization, Wasteful
Bubble Design

\[ Deg_{AC} = \sum_{i}^{N} (S_{AR_i} \times P_{CR_i}) \]
\[ Error = \sum_{i}^{N} |S_{AR_i} \times P_{BR_i} - S_{AR_i} \times P_{CR_i}| \]

A: app  
C: co-runner  
Bk: bubble at size k  
Ri: shared resource i

• Select k, closest to C  
• Error in proportions across Ri  
• Hypothesis: Prioritize first order  
  Rs and error will be small  
• Good bubble design is key  

• Many possible designs  
• Key Principles  
  • “Monotonic Curves”  
  • “Wide Dial Range”  
  • “Broad Impact”  
• Our bubble in the paper!
Our Bubble

- Uses “footprint” as pressure metric

- Three design points
  - Streaming activity (bandwidth)
  - Random accesses (caches)
  - Linear shift feedback register

```c
// Super cheap rand using a linear feedback shift register
#define MASK 0x00000001u
#define rand (lfsr = (lfsr >> 1) + (unsigned int)!
  (0 - (lfsr & 1u) & MASK))

unsigned int footprint_size = 0;
unsigned int dump[100];
#define r (rand%footprint_size)

while(1)
  { double *mid=bw_data+(bw_stream_size/2);
    for(int i=0; i<bw_stream_size/2; i++)
      { bw_data[i]=scalar*mid[i];
       }
    for(int i=0; i<bw_stream_size/2; i++)
      { mid[i]=scalar*bw_data[i];
       }
  }

while(1)
  { dump[0] += data_chunk[r]++;
    dump[1] += data_chunk[r]++;
    ...
    dump[98] += data_chunk[r]++;
    dump[99] += data_chunk[r]++;
  }
```
Defining QoS with Tolerance

- Changing how QoS is defined to include a range, reduces violations
- Useful in cases where contractual service level agreements are defined
Impact at Application Level

- Notice some applications loose
- However with MCs very little lost for big gains
- Much more results and insights in the paper, but here’s a sample...
4.3.3 Latency

In addition to the aggregated IPS, we also compare the latency of all jobs in a datacenter, defined as the execution time of the longest-running job under a given job-to-machine mapping. Figure 4.5 shows the latency of various mapping policies, normalized to the latency when all jobs run alone on their best performing machine type. Interestingly, although the random mapping can improve the average IPS performance, it performs equally poorly as the worst mapping for improving latency. In this experiment, SmartyMap improves the job placement of the slowest job resulting in lower overall latency. Again, Smarty-MCs performs the best, and Smarty-M performs comparably well.

4.3.4 Impact at the Application-level

We further examine the performance improvement of each application and compare it with the estimation by the opportunistic factor (Section 3.2). Figure 4.6 presents the performance improvement at the application level from the 2-Jobs scenario with 500 machines and 1000 applications for Google testbed as in Figures 4.4 and 4.5.

The y-axis shows each application type's performance using SmartyMap, normalized to each type's average performance.
Impact at Application Level

• Notice some applications loose

• However with SM-MCs very little lost for big gains

• Much more results and insights in the paper, but here’s a sample...
The Oblivion of Interference

- Performance of Search Render as Co-Runner Changes
- Latency-sensitive (user facing) workloads must attain high QoS
- Uncertainty leads to Over-provisioning
- And ultimately, low utilization
Adapting to Diverse EE

• Class of problems requiring online adaptation of applications to diverse EE
Adapting to Diverse EE

- Class of problems requiring online adaptation of applications to diverse EE
- Two real problems in modern WSCs
  - Cannot detect effectiveness of aggressive optimizations
  - Cannot detect when contention degrades performance
Adapting to Diverse EE

- Class of problems requiring online adaptation of applications to diverse EE
- Two real problems in modern WSCs
  - Cannot detect effectiveness of aggressive optimizations
  - Cannot detect when contention degrades performance
- Before we can solve problems, a mechanism is needed
Expensive and Inefficient
Expensive and Inefficient

- Expensive: $100s of Millions

In this example, which is typical of classical datacenters, the high server capital costs dominate overall TCO, with 69% of the monthly cost related to server purchase and maintenance. However, commodity-based lower-cost (and perhaps lower-reliability) servers, or higher power prices, can change the picture quite dramatically. For case B (see Figure 6.2), we assume a cheaper, faster, higher-powered server consuming 500 W at peak and costing only $2,000 in a location where electricity cost is $0.10/kWh. In this case, datacenter-related costs rise to 49% of the total, and energy costs to 22%, with server costs falling to 29%. In other words, the hosting cost of such a server, that is, the cost of all infrastructure and power to house it, is more than twice the cost of purchasing and maintaining the server in this scenario.

Note that even with the assumed higher power price and higher server power, the absolute 3-year TCO in case B is lower than in case A ($8,702 vs $10,757) because the server is so much cheaper. The relative importance of power-related costs is likely to increase as shown in case B.
Expensive and Inefficient

- Expensive: $100s of Millions

- Inefficient: Low Utilization
Expensive and Inefficient

- Expensive: $100s of Millions

- Inefficient: Low Utilization
Expensive and Inefficient

- Expensive: $100s of Millions
- Inefficient: Low Utilization

“...software performance and server utilization matter just as much [as hardware costs].”

-Barroso and Holzle
“...software **performance** and server **utilization** matter just as much [as hardware costs].”

-Barroso and Holzle
Reflecting on WSC Efficiency

<table>
<thead>
<tr>
<th>First Order Objective</th>
<th>Second Order Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Reflecting on WSC Efficiency

<table>
<thead>
<tr>
<th>First Order Objective</th>
<th>Second Order Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td></td>
</tr>
</tbody>
</table>
## Reflecting on WSC Efficiency

<table>
<thead>
<tr>
<th>First Order Objective</th>
<th>Second Order Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Functionality</strong></td>
<td></td>
</tr>
<tr>
<td>• Goals: A working, scalable system</td>
<td></td>
</tr>
<tr>
<td>• Uses general purpose components</td>
<td></td>
</tr>
<tr>
<td>• Architectures</td>
<td></td>
</tr>
<tr>
<td>• Commodity chips (Core 2 / Core i7 / K10)</td>
<td></td>
</tr>
<tr>
<td>• System software and tools</td>
<td></td>
</tr>
<tr>
<td>• Linux</td>
<td></td>
</tr>
<tr>
<td>• GCC, Binutils, JVM</td>
<td></td>
</tr>
</tbody>
</table>
Reflecting on WSC Efficiency

First Order Objective

- Goals: A working, scalable system
- Uses general purpose components
  - Architectures
    - Commodity chips (Core 2 / Core i7 / K10)
  - System software and tools
    - Linux
    - GCC, Binutils, JVM

Second Order Objective

- Functionality
- Efficiency
Reflecting on WSC Efficiency

**First Order Objective**

- Goals: A working, scalable system
- Uses general purpose components
  - Architectures
    - Commodity chips (Core 2 / Core i7 / K10)
  - System software and tools
    - Linux
    - GCC, Binutils, JVM

**Second Order Objective**

- Goals: better performance, higher utilization
- Tweak / evolve general purpose components
- Optimize system software and tools
- Tuning
Rethinking WSC Architecture
Rethinking WSC Architecture

- Insight: Core characteristic unique to WSCs has been overlooked

Cluster Level

Machine Level
Rethinking WSC Architecture

• Insight: Core characteristic unique to WSCs has been overlooked
  • Diversity in Execution Environments

Cluster Level

Machine Level
Rethinking WSC Architecture

• Insight: Core characteristic unique to WSCs has been overlooked
  • Diversity in Execution Environments

Cluster Level

Machine Level

Gen 1 Xeon
0 Jobs

Gen 2 Xeon
1 Job

Opteron
3 Jobs
Rethinking WSC Architecture

• Insight: Core characteristic unique to WSCs has been overlooked
  • Diversity in Execution Environments
Rethinking WSC Architecture

- Insight: Core characteristic unique to WSCs has been overlooked
  - Diversity in Execution Environments

![Diagram of Job Mapper and Execution Environments]

- Job Mapper
- Cluster Level
  - Gen 1 Xeon: 0 Jobs
  - Gen 2 Xeon: 1 Job
  - Opteron: 3 Jobs
- Machine Level
Rethinking WSC Architecture

- Insight: Core characteristic unique to WSCs has been overlooked
  - Diversity in Execution Environments

- Acknowledging diversity in EE is critical for efficiency
Rethinking WSC Architecture

• Insight: Core characteristic unique to WSCs has been overlooked
  • Diversity in Execution Environments

• Acknowledging diversity in EE is critical for efficiency
• How can we integrate the notion of diverse EE into how we design WSCs?
The Folly of the Homogenous Assumption

<table>
<thead>
<tr>
<th>CPU</th>
<th>GHz</th>
<th>Cores</th>
<th>L2/L3</th>
<th>Nickname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clovertown Xeon E5545</td>
<td>2.33ghz</td>
<td>6</td>
<td>8mb</td>
<td>Clover</td>
</tr>
<tr>
<td>Istanbul Opteron 8431</td>
<td>2.4ghz</td>
<td>6</td>
<td>6mb</td>
<td>Istan</td>
</tr>
<tr>
<td>Westmere Xeon X5660</td>
<td>2.8ghz</td>
<td>6</td>
<td>12mb</td>
<td>West</td>
</tr>
</tbody>
</table>

- Simple experimentation illustrates the folly
- Google workloads across various EE
- Job performance heavily impacted by placement
  - Microarchitectural configuration / co-runners

![Graph showing normalized performance across different applications and configurations]

![Graph showing performance impact across different configurations and placements]

Figure 3.1: These applications cover nine large industry-strength Google applications running on 3 types of production machines.

Chapter 3. Execution Milieus in WSCs

Finally, we introduce a metric, which is configured to three for both solo and co-location runs.

Each cluster is normalized to poorest performing architecture. Each cluster is composed of a host of Google workloads and machine clusters that have been both laboriously configured by a team of engineers for performance improvement from exploiting the heterogeneity in that platform is almost always better than the other two, the performance sensitivity of each application is to varying platform types. As we will discuss later, among the three for each application.

Performance variability, our study also aims to investigate firstly whether architectures. Each cluster is normalized to poorest performing architecture. Each cluster is composed of a host of Google workloads and machine clusters that have been both laboriously configured by a team of engineers for performance improvement from exploiting the heterogeneity in WSCs. In addition to quantifying the magnitude of that heterogeneity in WSCs. In addition to quantifying the magnitude of that heterogeneity in WSCs. In addition to quantifying the magnitude of that heterogeneity in WSCs.
My Dissertation Work
My Dissertation Work

Intelligently place jobs for diverse EEs

Job Mapper

- Gen 1 Xeon: 0 Jobs
- Gen 2 Xeon: 1 Job Running
- Opteron: 3 Jobs Running
My Dissertation Work

Intelligently place jobs for diverse EEs

Adapt jobs according to EE
My Dissertation Work

Intelligently place jobs for diverse EEs

Adapt jobs according to EE

Job Mapper

Gen 1 Xeon: 0 Jobs
Gen 2 Xeon: 1 Job Running
Opteron: 3 Jobs Running

Measure and Manage EE effects
My Dissertation Work

Opportunistic Mapping
CAL ’11

SmartyMap Mapping for Google
Under Submission @ ISCA ’12

Bubble Up
MICRO ’11, TopPicks ’12

Job Mapper

Gen 1 Xeon
0 Jobs

Gen 2 Xeon
1 Job Running

Opteron
3 Jobs Running

Characterization
ISCA ’11, MICRO ’11, ISPASS ’12

Bubble Up
MICRO ’11, TopPicks ’12

CiPE Methodology
HiPEAC ’11

Application Adaptation
CGO ’09, CGO ’10, EXADAPT ’11

Thread-to-Core Mapping
ISCA ’11, ISPASS ’12

Compiling for Contention
CGO ’12

Tuesday, October 8, 13
My Dissertation Work

Opportunistic Mapping
CAL '11

SmartyMap Mapping for Google
Under Submission @ ISCA '12

Bubble Up
MICRO '11, TopPicks '12

Job Mapper

Gen 1 Xeon
0 Jobs

Gen 2 Xeon
1 Job Running

Opteron
3 Jobs Running

Application Adaptation
CGO '09, CGO '10, EXADAPT '11

Thread-to-Core Mapping
ISCA '11, ISPASS '12

Compiling for Contention
CGO '12

Characterization
ISCA '11, MICRO '11, ISPASS '12

Bubble Up
MICRO '11, TopPicks '12

CiPE Methodology
HiPEAC '11

Tuesday, October 8, 13
Lightweight Online Adaption

Loaf Framework

Adaptation policy

Loaf SBM Compiler
SBM Versioning

Code Ninja

App

Adaptation Policy
Loaf Introspection Engine + API

App
CCAC
SCAC

1
2
3
...
Lightweight Online Adaption

- **Loaf**: Lightweight online adaptation framework

Loaf Framework

- Code Ninja
- Adaptation policy
- Loaf SBM Compiler
- SBM Versioning
- App
- Adaptation Policy
- CCAC
- Loaf Introspection Engine + API
- App
- App
- App
- CCAC
- CCAC
Lightweight Online Adaption

- **Loaf**: Lightweight online adaptation framework
- No fine grain control (avoids overhead)
Lightweight Online Adaptation

- **Loaf**: Lightweight online adaptation framework
  - No fine grain control (avoids overhead)
  - No/negligible instrumentation
Lightweight Online Adaption

- **Loaf**: Lightweight online adaptation framework
  - No fine grain control (avoids overhead)
  - No/negligible instrumentation
  - Must consider required capabilities for online adaptation
Lightweight Online Adaption

- **Loaf**: Lightweight online adaptation framework
  - No fine grain control (avoids overhead)
  - No/negligible instrumentation
  - Must consider required capabilities for online adaptation

  1. *Lightweight* online introspection and monitoring
Lightweight Online Adaption

- **Loaf**: Lightweight online adaptation framework
  - No fine grain control (avoids overhead)
  - No/negligible instrumentation
  - Must consider required capabilities for online adaptation

1. **Lightweight** online introspection and monitoring
2. **Lightweight** online application code adaptation
Lightweight Online Adaption

- **Loaf**: Lightweight online adaptation framework
  - No fine grain control (avoids overhead)
  - No/negligible instrumentation
  - Must consider required capabilities for online adaptation

1. **Lightweight** online introspection and monitoring
2. **Lightweight** online application code adaptation
3. **Lightweight** online application environment adaptation
Lightweight Online Adaption

Adaptation policy

Code Ninja

Loaf SBM Compiler
SBM Versioning

App

Adaptation Policy
Loaf Introspection Engine + API

App
CCAC

App
CCAC
Lightweight Online Adaptation Framework (Loaf)

1. Code Ninja
   - Adaptation Policy

2. Loaf SBM Compiler
   - Loaf SBM Versioning

3. App
   - Adaptation Policy
   - Loaf Introspection Engine + API

App
   - CCAC
   - CCAC
   - ...
Lightweight Online Adaptation Framework (Loaf)

Adaptation policy

Loaf SBM Compiler
SBM Versioning

Code Ninja

Lightweight monitoring

Job Mapper
Gen 1 Xeon
0 Jobs
Gen 2 Xeon
1 Job Running
Opteron
3 Jobs Running

App
Adaptation Policy
Loaf Introspection
Engine + API

App
App
App

CCAC
CCAC
CCAC

Tuesday, October 8, 13
Lightweight Online Adaptation Framework (Loaf)

Adaptation policy

Code Ninja

Loaf SBM Compiler

SBM Versioning

2

Code Adaptation

App

Adaptation Policy

Loaf Introspection Engine + API

Lightweight monitoring

1

App

CCAC

App

CCAC

...
Lightweight Online Adaptation Framework (Loaf)

Adaptation policy

Code Ninja

Lightweight monitoring

Loaf SBM Compiler

SBM Versioning

2

Code Adaptation

Cooperative Adaptation

App

Adaptation Policy

Loaf Introspection Engine + API

1

Job Mapper

Gen 1 Xeon

0 Jobs

Gen 2 Xeon

1 Job Running

Opteron

3 Jobs Running

...
Lightweight Online Adaptation Framework (Loaf)

Adaptation policy

Loaf SBM Compiler

SBM Versioning

Code Ninja

App

Adaptation Policy

Loaf Introspection Engine + API

CCAC

App

CCAC

App

CCAC

...
Challenges for Application Adaptation
Challenges for Application Adaptation

• Traditional techniques not suitable to WSCs
  • Managed runtimes -> sacrifice performance, managed
  • Dynamic binary translators -> huge complexity, sacrifice performance
    • Not adopted in production
Challenges for Application Adaptation

- Traditional techniques not suitable to WSCs
  - Managed runtimes -> sacrifice performance, managed
  - Dynamic binary translators -> huge complexity, sacrifice performance
    - Not adopted in production
- To achieve deployability within WSCs
  - Must be lightweight
  - Low complexity
Challenges for Application Adaptation

• Traditional techniques not suitable to WSCs
  • Managed runtimes -> sacrifice performance, managed
  • Dynamic binary translators -> huge complexity, sacrifice performance
    • Not adopted in production
• To achieve deployability within WSCs
  • Must be lightweight
  • Low complexity
• We need new technology for online adaptation
Challenges for Application Adaptation

- Traditional techniques not suitable to WSCs
  - Managed runtimes -> sacrifice performance, managed
  - Dynamic binary translators -> huge complexity, sacrifice performance
    - Not adopted in production
- To achieve deployability within WSCs
  - Must be lightweight
  - Low complexity
- We need new technology for online adaptation
- This new technology is one of the major challenges and contributions of this work