Final Thesis

A Study of Scalability and Performance of Solaris Zones

by

Yuan Xu

LITH-IDABEX--07/010--SE

2007-03-12
Final Thesis

A Study of Scalability and Performance of Solaris Zones

by

Yuan Xu

LITH-IDA-EX--07/010--SE

2007-03-12

Supervisor: Prof. Dr. Christoph Schuba
Examiner: Prof. Dr. Christoph Schuba
**Title**

An Study of Scalability and Performance of Solaris Zones

**Abstract**

This thesis presents a quantitative evaluation of an operating system virtualization technology known as Solaris Containers or Solaris Zones, with a special emphasis on measuring the influence of a security technology known as Solaris Trusted Extensions. Solaris Zones is an operating system-level (OS-level) virtualization technology embedded in the Solaris OS that primarily provides containment of processes within the abstraction of a complete operating system environment. Solaris Trusted Extensions presents a specific configuration of the Solaris operating system that is designed to offer multi-level security functionality.

Firstly, we examine the scalability of the OS with respect to an increasing number of zones. Secondly, we evaluate the performance of zones in three scenarios. In the first scenario we measure - as a baseline - the performance of Solaris Zones on a 2-CPU core machine in the standard configuration that is distributed as part of the Solaris OS. In the second scenario we investigate the influence of the number of CPU cores. In the third scenario we evaluate the performance in the presence of a security configuration known as Solaris Trusted Extensions. To evaluate performance, we calculate a number of metrics using the AIM benchmark. We calculate these benchmarks for the global zone, a non-global zone, and increasing numbers of concurrently running non-global zones. We aggregate the results of the latter to compare aggregate system performance against single zone performance.

The results of this study demonstrate the scalability and performance impact of Solaris Zones in the Solaris OS. On our chosen hardware platform, Solaris Zones scales to about 110 zones within a short creation time (i.e., less than 13 minutes per zone for installation, configuration, and boot.) As the number of zones increases, the measured overhead of virtualization shows less than 2% of performance decrease for most measured benchmarks, with one exception: the benchmarks for memory and process management show that performance decreases of 5-12% (depending on the sub-benchmark) are typical. When evaluating the Trusted Extensions-based security configuration, additional small performance penalties were measured in the areas of Disk/Filesystem I/O and Inter Process Communication. Most benchmarks show that aggregate system performance is higher when distributing system load across multiple zones compared to running the same load in a single zone.

**Keywords**

Solaris Containers, Solaris Zones, Solaris Operating System, Solaris Trusted Extensions, scalability evaluation, performance evaluation, security metrics, AIM benchmark
Abstract

This thesis presents a quantitative evaluation of an operating system virtualization technology known as Solaris Containers or Solaris Zones, with a special emphasis on measuring the influence of a security technology known as Solaris Trusted Extensions. Solaris Zones is an operating system-level (OS-level) virtualization technology embedded in the Solaris OS that primarily provides containment of processes within the abstraction of a complete operating system environment. Solaris Trusted Extensions presents a specific configuration of the Solaris operating system that is designed to offer multi-level security functionality.

Firstly, we examine the scalability of the OS with respect to an increasing number of zones. Secondly, we evaluate the performance of zones in three scenarios. In the first scenario we measure - as a baseline - the performance of Solaris Zones on a 2-CPU core machine in the standard configuration that is distributed as part of the Solaris OS. In the second scenario we investigate the influence of the number of CPU cores. In the third scenario we evaluate the performance in the presence of a security configuration known as Solaris Trusted Extensions. To evaluate performance, we calculate a number of metrics using the AIM benchmark. We calculate these benchmarks for the global zone, a non-global zone, and increasing numbers of concurrently running non-global zones. We aggregate the results of the latter to compare aggregate system performance against single zone performance.

The results of this study demonstrate the scalability and performance impact of Solaris Zones in the Solaris OS. On our chosen hardware platform, Solaris Zones scales to about 110 zones within a short creation time (i.e., less than 13
As the number of zones increases, the measured overhead of virtualization shows less than 2% of performance decrease for most measured benchmarks, with one exception: the benchmarks for memory and process management show that performance decreases of up to 5-12% (depending on the sub-benchmark) can occur. When evaluating the Trusted Extensions-based security configuration, additional small performance penalties were measured in the areas of Disk/Filesystem I/O and Inter Process Communication. Most benchmarks show that aggregate system performance is higher when distributing system load across multiple zones compared to running the same load in a single zone.

**Keywords:**
Solaris Containers, Solaris Zones, Solaris Operating System, Solaris Trusted Extensions, scalability evaluation, performance evaluation, security metrics, AIM benchmark
Acknowledgments

First and foremost, I wish in these lines to thank everyone. Grateful thanks are due to Prof. Dr. Christoph Schuba for constructive comments and valuable suggestions during my study at Linköping University.

I also wish to thank all my friends in Linköping for always making me feel welcome and feel less far away from home. Thank you for the good time we had together. Everybody including my friends in China, in his or her own way, has helped me through the difficult situations that I experienced.

Last but not least, I would like to thank my wonderful family, including my parents and my boyfriend, for all their enduring support and always believing in me.
# Table of Contents

Chapter 1  
Introduction ........................................................................................................ 1  
1.1 Background .................................................................................................... 1  
1.1.1 Overview of Solaris Zones ........................................................................ 2  
1.1.2 History and development ......................................................................... 4  
1.1.3 Benefits of Solaris Zones .......................................................................... 7  
1.1.4 Overview of Solaris Trusted Extensions .................................................... 8  
1.2 Motivation ...................................................................................................... 9  
1.3 Typographic Conventions ............................................................................ 10  
1.4 Roadmap ....................................................................................................... 11  

Chapter 2  
Problem Statement ............................................................................................ 13  

Chapter 3  
Related Research ................................................................................................ 15  
3.1 The Evaluations of Existing Virtualization Technologies ......................... 15  
3.1.1 Observation-based evaluation .................................................................. 16  
3.1.2 Responsiveness-based evaluation ............................................................... 17  
3.1.3 Quantitative evaluation ............................................................................. 18  
3.1.4 Summary .................................................................................................... 25  
3.2 Existing Evaluations of Solaris Zones ......................................................... 26  

Chapter 4  
Evaluation of Solaris Zones ............................................................................. 27  
4.1 Test Bed ......................................................................................................... 27  
4.1.1 System description .................................................................................... 27  
4.1.2 Specification of zone ................................................................................ 28  
4.2 Test Tool ........................................................................................................ 30  
4.2.1 Introduction ............................................................................................... 30  
4.2.2 Requirement ............................................................................................. 30  
4.2.3 Configuration ............................................................................................. 31  
4.2.4 Results ...................................................................................................... 33  
4.3 Test Protocol .................................................................................................. 35  
4.3.1 Scalability .................................................................................................. 35  
4.3.2 Performance ............................................................................................. 35  
4.4 Test Description ............................................................................................. 38  
4.4.1 Tests on adit2-105 ................................................................................... 38  
4.4.2 Tests on adit2-106 ................................................................................... 43
Chapter 5
Results and Analysis ................................................................. 47
  5.1 Scalability .................................................................................. 47
  5.2 Performance .............................................................................. 50
    5.2.1 Performance of running the benchmark in one zone ............ 50
    5.2.2 Performance of running benchmarks concurrently in different zones .... 62
  5.3 Discussion ................................................................................. 80
    5.3.1 Discussion based on system scalability .............................. 80
    5.3.2 Discussion based on system performance ...................... 81
  5.4 Summary ................................................................................. 83

Chapter 6
Conclusion and Further Research ........................................... 85
  6.1 Conclusion ............................................................................... 85
  6.2 Methodology Limitations ....................................................... 87
  6.3 Future Work ............................................................................ 88

Reference .................................................................................... 91

Appendix
  Scripts of evaluations ................................................................. 93
Chapter 1
Introduction

The Solaris operating system is a POSIX-compliant operating system (OS), developed by Sun Microsystems. It has become increasing popular, because the Solaris OS is the first commercially developed UNIX system that was open sourced. Solaris Zones\(^1\) is an implementation of an operating system-level virtualization technology first available in a mainstream operating system in Solaris 10. Zones in the Solaris OS act as completely isolated OS instances within one physical machine. Thus, this technology can contribute a good platform for the implementation of a variety of different security policies for an organization.

This chapter presents an introduction to Solaris Zones and to Solaris Trusted Extensions, its technology for security policy enforcement. Then, the motivation and importance of implementing the performance evaluation are described, as well as the structure of this thesis.

1.1 Background

The Solaris Zones facility provides an isolated environment to run applications

\(^1\)A similar technology has been available in Trusted Solaris for many years.
in the Solaris Operating System. It builds on and extends the concept of a virtual operating system environment by including many of the features of a separate machine. The detailed introduction to Zones and to Solaris Trusted Extensions will be presented in the following subsections: overview, history, benefits, and Solaris Trusted Extensions.

1.1.1 Overview of Solaris Zones

Solaris is a computer operating system developed by Sun Microsystems. It is certified against the POSIX Interface as a version of UNIX. Although Solaris is still proprietary software, its core OS has been made into an open source project, called OpenSolaris. The underlying Solaris code base has been managed under continuous development since work began in the early 1980s and it was eventually released as Solaris 2.0. Each version such as Solaris 10 is based on a snapshot of this development "train", taken near the time of its release, which is then maintained as a derived project. Updates to this project are built and delivered several times a year until the next official release comes out. The latest version of Solaris is Solaris 10 update 4 which was released on January 2007. The tested system of our evaluation is running on SunOS 5.11 snv_44, which is still being updated biweekly, but is intended to be released as Solaris 11.

The Solaris Zones partitioning technology can be used on any machine that is running Solaris 10 or newer releases. A zone is a virtualized operating system environment which allows the applications in it to run isolated from the rest of the system. This isolation prevents processes running in one zone from monitoring or affecting processes running in other zones. Besides, a zone also provides an abstract layer that separates applications from the physical attributes of the machine on which they are deployed, such as physical device paths.
Figure 1-1: Solaris Zones facility

There are two types of zones for the Solaris Zones mechanism. As shown in Figure 1-1, the Solaris Operating System runs typically directly on the hardware, it manages the boot process, and initializes interfaces to the CPUs, memory, host bus adapters, network interface cards (NICs), storage, and device drivers in the system. Only one instance of the Solaris operating system runs on the hardware, and it is referred to as the global zone. The administrator creates non-global zones which exist above the global zone and access the hardware through the global zone. A non-global zone appears to all users — end users, applications, developers, and the zone administrator — as a fully realized OS instance with its own host name, IP address, process and name space, root and user names and passwords, network devices and file systems.
1.1.2 History and development

There is a tendency that increasing numbers of people are concerned about the way how to optimize the utilization of their computing resources. Since the late 1960’s, technologies for this purpose have been in development, which consists of running multiple applications and even multiple operating systems on the same hardware. As shown in Figure 1-2, the previous work in this area solves the problem of resource allocation from two main aspects: running multiple operating system instances and virtualizing the operating system environment.

Figure 1-2: Catalog and instances of OS virtualization technologies
Multiple operating system instances

Many of the previous projects have implemented multiple operating system instances on a single machine to distribute their computing resources. Generally, logical partitioning and virtual machine monitors are the two major strategies. Logical partitioning is the physical division of a computer's processors, memory, and storage into multiple sets of resources so that each set of resources can be operated independently with its own operating system instance and applications. It is a good strategy to provide a very high degree of application isolation, increase flexibility of resource allocation, as well as to achieve better hardware utilization. It was first studied by IBM in 1976 and now both IBM's S/390 (now z/900 Series) and AS/400 products support logical partitioning.

A virtual machine monitor, commonly called hypervisor, is a low-level software layer that allows multiple operating systems to run on a computer at the same time. Thus, it can be much more granular in resource allocation, even in the case of time-sharing multiple VM’s on a single CPU. The concept of a hypervisor was originally developed in the early 1970’s with the design of the IBM S/360 Model 67 mainframe computer. The commercial product VMware ([9]) was introduced in the mid-1990’s for the Intel x86 instruction-set machines. Today, Xen ([1]) and TRANGO ([5]) are implemented as software-only virtual machines. Xen focuses on normal host operating systems, while TRANGO pays more attention to embedded systems.

Virtualizing the operating system environment

By early 2007, a lot of projects have been focusing on virtualizing operating system environments, such as FreeBSD Jail ([6]), Linux-VServer ([9]), and Virtuozzo ([5]). Although there is only one underlying operating system kernel, this concept has the ability to run multiple applications in isolation from each
other. The basic strategy can briefly be described as running groups of processes that cannot be interrupted by others in different virtual environments.

The FreeBSD jail mechanism is an early implementation of operating system-level virtualization. It allows administrators to partition a FreeBSD-based computer system into several independent mini-systems called jails. Rather than introducing an additional fine-grained access control mechanism, the solution adopted was to compartmentalize the system, including processes, file system, and network resources, so that they can only be accessed by the right compartment. Besides, processes within a jail are unable to interact or even verify the existence of processes or resources outside their jail. From the point of view of a process within a jail, its environment within the jail is almost indistinguishable from a real system.

The advantages of FreeBSD jails can be summarized by the following three aspects: virtualization, security and ease of delegation. Because each jail is a virtual environment running on a host machine with its own files, processes, user and superuser accounts, it is sealed off from others, thus providing an additional level of security. In addition, administrators can delegate tasks that require superuser access without handing out complete control over the system for the limited scope of a jail.

Linux-VServer is a virtual private server implementation done by adding operating system-level virtualization capabilities to the Linux kernel. Each partition for Linux-VServer is called a security context, and the virtualized system within it is the virtual private server. Virtual private servers are commonly used in web hosting services, where they are useful for segregating customer accounts, pooling resources and resisting any potential security breaches. Besides, Linux-VServer uses the jail mechanism which securely partitions resources on a computer system, such as the file system, CPU time,
network addresses and memory. Thus, processes cannot mount a denial-of-service attack on anything outside their partition.

Virtuozzo is a proprietary operating system virtualization product produced by SWsoft, Inc. It can create multiple isolated virtual environments (virtual private servers) on a single physical server. It can support tens to hundreds of virtual private servers on a single server due to its use of operating system-level virtualization and it is available for both Linux and Microsoft Windows.

Solaris Zones is built on the same technology, operating system-level virtualization. However, it extends its virtual operating system environment to include many more features of a separate machine, such as a per-zone console, system log, packaging database, run level, identity (including name services), and interprocess communication facility.

1.1.3 Benefits of Solaris Zones

Zones act as completely isolated virtual machines within a computer and reduce cost in both hardware and system administration. Furthermore, the Solaris Zones mechanism can provide protection through compartmentalization for separate virtual machines on a single physical machine.

Compared to the previously presented strategies, Solaris Zones has additional improvements. It is cheaper to install and to configure, because only a single copy of OS is involved, compared to several OS instances in the case of Xen. Furthermore, it is not limited for high-end systems compared to logical partitioning. Moreover, the granularity of resource allocation is fine-grained than the logical partitioning. In comparison to virtual machine monitors, Solaris Zones reduces performance overheads and reduces the cost of administration because there are no multiple operating system instances in a system.
A set of administrative tools has been developed to manage zones so that zones can be administered similar to how separate machines are administered. Besides, Zones provide the delegation of administrative controls for the virtual operating system environment, for example, each zone has its own name service identity, its own password file and its own root user. This characteristic gives an opportunity to define the maximum proportion of shared resources. In the case that one zone is successfully compromised, the compartmentalization aspect of Solaris Zones prevents the spreading of the damage to other zones.

**1.1.4 Overview of Solaris Trusted Extensions**

Solaris Trusted Extensions is an optionally-enabled layer of secure labeling technology that allows data security policies to be separated from data ownership. This new approach allows the Solaris operating system to support both traditional Discretionary Access Control (DAC) policies based on file or device ownership, as well as label-based Mandatory Access Control (MAC) policies.

The label-based MAC policies are implemented by adding labels to Solaris Zones. Each zone is assigned a unique sensitivity label and can be customized with its own set of file systems and network resources. When Solaris Trusted Extensions is enabled, all data flows are restricted based on a comparison of the labels associated with the subjects requesting access and the objects containing the data. These features enable an organization to define and implement a security policy on a Solaris system by defining differently labeled zones.

The four major security features that Trusted Extensions provides can be summarized as the following: protecting against intruders, providing mandatory access control, separating information by label, and enabling secure administration. Its MAC policy is enforced automatically and transparently.
without requiring customized applications or complex application profiles that are difficult to deploy and to maintain. Its administrative roles with discrete capabilities administer the system. In conclusion, the Solaris Trusted Extensions mechanism makes it possible to deliver a cost-effective and interoperable system that simultaneously provides ease-of-use and a high level of information assurance.

1.2 Motivation

Computers and the Internet have become an almost indispensable commodity in many people’s lives, like clothes, shelter, and even food. At work, people use computers to solve issues in their office, while at home they spend their time surfing the Internet, playing games, and so on. Altogether, there is a tendency that the world of human beings has been influenced by these high technology products.

As a result, more and more projects and companies devote themselves to develop products according to the demands of customers or problems in progress. Because of the interest in utilization of computing resources, the operating system-level virtualization mechanism Solaris Zones has been released with Solaris 10 in 2005. Since then, some projects have compared Zones with other relevant products and tested if Zones can deliver on its promise. However, few evaluations focus on its performance impact on the operating system. We consider this study to be an important factor to evaluate the usability of a new product.

This thesis measures and evaluates the performance impacts of Solaris Zones on the Solaris OS under different system configurations. The evaluation is performed in two parts. The first part is to examine how well the Solaris OS scales with respect to an increasing number of zones. The second part is to
measure the performance influence in three scenarios. In the first scenario we measure - as a baseline - the performance of Solaris Zones on a 2-CPU core machine in the standard configuration that is distributed as part of the Solaris OS. In the second scenario we investigate the influence of the number of CPU-cores. In the third scenario we evaluate the performance in the presence of a security configuration known as Solaris Trusted Extensions.

### 1.3 Typographic Conventions

Table 1-1 describes the typographic changes that are used in this thesis.

<table>
<thead>
<tr>
<th>Typeface or Symbol</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AaBbCc123</code></td>
<td>The names of commands and files</td>
<td>Use the <code>zonecfg</code> command Edit the file <code>s9workfile</code>.</td>
</tr>
<tr>
<td><code>AaBbCc123</code></td>
<td>What we type, contrasted with onscreen computer output</td>
<td>Type <code>-lnsl –socket</code>.</td>
</tr>
<tr>
<td><code>AaBbCc123</code></td>
<td>The content of script</td>
<td><code>#!/bin/bash</code> <code>for ((i=1; i&lt;=130; i++)) do</code></td>
</tr>
<tr>
<td><code>AaBbCc123</code></td>
<td>The name of sub-benchmarks</td>
<td><code>Int Additions</code></td>
</tr>
<tr>
<td><code>AaBbCc123</code></td>
<td>The group name of sub-benchmarks</td>
<td><code>InterProcess Communication</code></td>
</tr>
<tr>
<td><code>AaBbCc123</code></td>
<td>The name of the machine</td>
<td><code>Adit2-105</code></td>
</tr>
</tbody>
</table>

**Table 1-1: Typographic conventions for this thesis**

In a printed copy of this thesis, the figures and graphs are likely to appear black and white, which makes some of the figures and graphs difficult to interpret correctly. An electronic copy of this thesis, which contains these figures and graphs in color and high resolution, can be found at [http://www.ep.liu.se](http://www.ep.liu.se).
1.4 Roadmap

Chapter 2 assesses some evaluations of operating system-level virtualization mechanisms. Chapter 3 presents the research problems for this thesis. Chapter 4 describes the environment, tool and protocol of the evaluation and how it could be implemented to characterize the performance of Solaris Zones. Chapter 5 shows the results and analysis of performance impacts of Solaris Zones on the Solaris OS under different system configurations. Chapter 6 contains our conclusions and presents some ideas for further work.
Chapter 2

Problem Statement

In this chapter, we present the objective of our work and further map out the corresponding problems for the project. There is a tendency that increasing numbers of users not only care about whether any given virtualization technology can work or fulfill their functional requirements, but also about the performance impact on the corresponding OS.

The objective of our work is to measure and evaluate the performance impacts of Solaris Zones on the Solaris OS under different system configurations. Two research problems are proposed. The first research problem is to examine how well the Solaris OS scales with respect to an increasing number of zones. The second research problem is to measure the performance influence under different configurations.

- Scalability
  
  We examine how well the OS scales with respect to an increasing number of zones. Our analysis focuses on the usability of Solaris Zones, rather than detecting the maximum number of zones that the operating system can support. We look for the exact number of zones that the OS can support while still giving the user “acceptable” performance.
Performance

a) Baseline performance of using Solaris Zones
In this part we measure - as a baseline - the performance of Solaris Zones on a 2-CPU core machine in the standard configuration that is distributed as part of the Solaris OS. The performance in the global zone and in non-global zones will be evaluated as the number of zones increases. We also want to compare the performance of running applications concurrently in different zones to running them on the same native operating system.

b) Performance of running on one CPU core
We investigate the influence of the number of CPU cores by evaluating the performance with only one CPU core enabled.

c) Performance of security configuration enabled
The security mechanism we applied for Solaris Zones is made available through Solaris Trusted Extensions. Trusted Extensions enables the enforcement of mandatory access control (MAC) and discretionary access control (DAC) between communicating processes on the same host as well as across a network. As a result, the throughput of read and write between each labeled zone might be negatively affected, so weight be the throughput of interprocess communication. We plan to validate this expected reduction in performance compared to the baseline performance of Solaris Zones.
Chapter 3

Related Research

There are various different approaches to running virtual machines on a single machine. Their implementation principles are different from each other. Some approaches allow installing and running several operating system instances on a single machine. Others require virtualizing operating system environments on one underlying operating system kernel. However, all OS virtualization technologies have similar objectives, which can be summarized as the following: transparency, portability, VM performance, confinement and performance isolation.

In this chapter, we focus on approaches how to evaluate for these virtualization technologies, especially their scalability and performance. Published evaluation methods and their results for existing virtualization technologies and for Solaris Zones are presented in the two sections below.

3.1 The Evaluations of Existing Virtualization Technologies

Previous work we have studied falls into one of two categories. It either evaluates the performance of a specific virtualization mechanism, or if focuses
on the comparison of several different technologies. In this section, several
typical evaluation strategies are introduced, followed by a small summary of
evaluation methods.

3.1.1 Observation-based evaluation

In the paper “Virtualization of Linux based computers” ([9]), the author’s
purpose is to evaluate the Linux-VServer technology in the field of Linux
virtualization. The article presents the available programs that can provide
virtualization for linux computers, such as VMware, plex86, Bochs,
Linux-VServer, User Mode Linux, Xen and QEMU, as described in [9]. Then,
two tests are performed to confirm that the Linux-VServer is the best of them.
One test is applying some usage criteria for each mechanism, while the other is
doing a comparative study based on a number of use-cases.

For the first test, the author defines several requirements for computer
virtualization and uses those criteria to compare the selected technologies.
Table 3-1 presents the results of the different virtualization mechanisms.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>VMware</th>
<th>Linux-VServers</th>
<th>UML</th>
<th>Xen</th>
<th>QEMU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi OS</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>exp.</td>
</tr>
<tr>
<td>Kernel Development</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>exp.</td>
<td>no</td>
</tr>
<tr>
<td>Install Process</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>exp.</td>
</tr>
<tr>
<td>Resources</td>
<td>2Gb</td>
<td>256Mb</td>
<td>1Gb</td>
<td>1Gb</td>
<td>1Gb</td>
</tr>
<tr>
<td>Dynamical Resources</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Security</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Maturity</td>
<td>good</td>
<td>excellent</td>
<td>good</td>
<td>young</td>
<td>young</td>
</tr>
<tr>
<td>Architecture</td>
<td>x86</td>
<td>x86,IA64,x86_64</td>
<td>x86,IA64,x86_64</td>
<td>x86</td>
<td>x86</td>
</tr>
</tbody>
</table>

Table 3-1: Comparison of Linux-based virtualization technologies
according to [9]
The author establishes seven basic use-cases to illustrate the Linux-VServer is the best suited technique for them. The sample use-cases for using Linux-VServer mechanism are listed as the following:

- Hosting: an Internet provider or someone who has to provide access to one or several hosts.
- Testing one application: make stress tests or unitary testing on one application.
- Building or developing environment.
- Testing distributed applications and/or complex upgrade processes.
- Considering security usage
- Providing high Availability
- Offering disaster recovery

As a result, the Linux-VServer is considered by the author to be the best technology for virtualizing Linux servers. Compared with other virtualization techniques, the key advantages of the Linux-VServer technology is its small performance overhead.

3.1.2 Responsiveness-based evaluation

The authors of the paper ([8]) describe a Grid service VMPlant that provides automated configuration and creation of flexible VMs. An evaluation of its performance is presented including the analysis of creation latencies and cloning time.

An end-to-end VM creation time for the machines under evaluation with different memory sizes is shown in Figure 2-1(a). The normalized occurrence distribution (y-axis) is obtained from the creation time recorded for each VM successfully created. Two important observations can be drawn from these results: first, the DAG-based configuration and the cloning mechanism allow
VMs to be instantiated, on average, in 25 to 48 seconds. Second, the results show that creation times are larger for machines with larger memory sizes.

The distribution summarized in Figure 3-1(b) shows the profile of cloning time versus the order in which VM clones are requested. The cloning times tend to increase when the VMPlant hosts a large number of VMs, especially in the 64MB and 256MB cases.

![Graphs](image)

(a) VM creation latencies  
(b) Cloning time as a function of sequence number

**Figure 3-1: Distribution of VM creation latencies and clone time for VMPlants**

The authors of the paper “Dynamic Virtual Environments in the Grid” ([7]) model dynamic virtual environments (DVEs) as first-class entities in a distributed environment, with Grid service interfaces defined to negotiate creation, monitor properties, and manage lifetime. For their evaluation, they compared create/destroy times directly over the operating system, under VServer, and under VMware.

### 3.1.3 Quantitative evaluation

In this section, three well-organized quantitative evaluations will be introduced. They are scalability comparisons of 4 host virtualization tools ([11]), scale and performance in the Denali isolation kernel ([16]), and the evaluation of Xen ([1]).
3.1.3.1 Scalability comparisons of 4 host virtualization tools

In paper ([11]), the authors compared the scalability merits of 4 virtualization tools (Vserver, Xen, UML and VMware). The evaluation was implemented from two main perspectives: usability and performance.

Some usability criteria restrict the number of simultaneously running VMs, such as startup time and memory occupation. Startup time is an important characteristic of a virtualization technology, especially in a system where there are frequent reconfigurations. It is a part of the responsiveness of the virtualization technology. They demonstrate that Vserver outperforms Xen and UML in the case of cold and hot concurrent startup time. Besides, memory footprint can be measured by recording the physical memory footprint before the execution of the virtualization system and after that.

For the performance part, the authors focus on metrics related to machine virtualization performance. A virtualization technology exhibits a low constant virtualization overhead and allows performance isolation between virtualized machines independent of the number of running virtual machines. Thus, four sub-benchmarks are applied to measure the different scalability parameters for the different machine resources (CPU, memory disk and network) on three scalability metrics (overhead, linearity and isolation). The overhead can be calculated as the remainder of the execution time for an application running on a virtualized OS and on a non-virtualized one. Linearity is to evaluate changes in the performance as the number of running virtual machines is increased. It is tested by comparing the results with its theoretical value. In addition, performance isolation ensures that in a situation of load imbalance between VMs, all VMs will still get an equal access to the machine resources. Figure 3-2 presents three scalability metrics on CPU according to the number of concurrent VMs running the application. Finally, the authors test the
communications between virtual machines, which gives the bandwidth available for every VM.

Figure 3-2: CPU overhead, CPU linearity and CPU performance isolation

3.1.3.2 Scale and performance in the Denali isolation kernel

In this paper, the authors describe the Denali isolation kernel ([16]), an operating system architecture that safely multiplexes a large number of untrusted Internet services on shared hardware. A quantitative evaluation of Denali is presented to quantify the performance of Denali's primitive operations, validating that the virtual architecture modifications result in enhanced scale, performance, and simplicity, and characterize how the system performs at scale.

To characterize Denali's performance, the authors measured the context switching overhead between VMs, and the swap disk subsystem performance. They also characterized virtualization overhead by analyzing packet dispatch
latency, and by comparing the application-level TCP and HTTP throughput of Denali with that of BSD.

To provide quantitative evidence about enhanced scalability, performance, and simplicity of Denali, the authors demonstrate that batched asynchronous interrupts have performance and scalability benefits (Figure 3-3(a)). Besides, Denali's idle-with-timeout instruction is crucial for scalability (Figure 3-3(b)). Finally, Denali's simplified virtual NIC has performance advantages over an emulated real NIC, and that the source code complexity of Denali is substantially less than that of even a minimal Linux kernel (Figure 3-4).

Figure 3-3: Benefits for interrupts and Idle-with-timeout

Figure 3-4: Source code complexity
To characterize Denali’s performance at scale, the authors first analyze two scaling bottlenecks, and then evaluate two applications with fairly different performance requirements and characteristics: a web server and the Quake II game server.

- **Scaling bottlenecks**
  The number of virtual machines to which our isolation kernel can scale is limited by two factors: per machine metadata maintained by the kernel when a VM has been completely paged out, and the working set size of active VMs.

- **Web server performance**
  To understand the factors that influence scalability for a throughput-centric workload, the authors analyzed Denali’s performance when running many web server VMs. Three factors strongly influenced scalability: disk transfer block size, the popularity distribution of requests across VMs, and the object size transferred by each web server.

- **Quake II game server performance**
  Because Quake II is a latency-sensitive multiplayer Game, the authors use two metrics as a measure of the quality of the game experience: the latency between a client sending an update to the server and receiving back a causally dependent update, and the throughput of updates sent from the server. Steady latency and throughput are necessary for a smooth, lag-free game experience.

### 3.1.3.3 An evaluation of Xen

The paper “Xen and the Art of Virtualization” ([1]) presents a thorough performance evaluation of Xen. The authors begin by benchmarking Xen against a number of alternative virtualization techniques, and then compare the total system throughput executing multiple applications concurrently on a single native operating system against running each application in its own
virtual machine. They then evaluate the performance isolation Xen provides between guest OSes, and assess the total overhead of running large numbers of operating systems on the same hardware.

Firstly, they have performed a battery of experiments in order to evaluate the overhead of the various virtualization techniques, including native Linux, XenoLinux, VMware workstation 3.2 and User-mode Linux. The six experiments consist of SPEC CPU2000 suite, the total elapsed time taken to build a default configuration of the Linux 2.4.21 kernel, the multi-user Information Retrieval (IR) and On-Line Transaction Processing (OLTP) workloads of the Open Source Database Benchmark suite (OSDB), a file system benchmark dbench, and SPEC WEB99 suite (Figure 3-5).

![Figure 3-5: Relative performance of native Linux (L), XenoLinux (X), VMware (V) and User-Mode Linux (U)](image)

To more precisely measure the areas of overhead within Xen and the other VMMs, the authors performed a number of smaller experiments targeting particular subsystems by McVoy’s lmbench program ([1]). They present the results from four aspects: processes times, context switching times, file system latencies, and bandwidth.
Then, the authors compare the performance of running multiple applications in their own guest OS against running them on the same native operating system. Figure 3-6 shows the results of running 1, 2, 4, 8 and 16 copies of the SPEC WEB99 benchmark in parallel, and the aggregate throughput Xen achieves when running 1, 2, 4 and 8 instances of OSDB-IR and OSDB-OLTP.

In order to demonstrate the performance isolation provided by Xen, the authors ran 4 domains configured with equal resource allocations, with two domains running previously-measured workloads (PostgreSQL/OSDB-IR and SPEC WEB99), and two domains each running a pair of extremely antisocial processes. They found that both the OSDB-IR and SPEC WEB99 results were only marginally affected by the behavior of the two domains running disruptive processes. When repeating the same experiment under VMware workstation and native Linux, VMware Workstation achieves similar levels of isolation, but at reduced levels of absolute performance, while the disruptive processes rendered the native Linux completely unusable for the two benchmark processes, causing almost all the CPU time to be spent in the OS.

Finally, they examine Xen’s ability to scale to its target of 100 domains. They discuss the memory requirements of running many instances of a guest OS and
associated applications, and measure the CPU performance overhead of their execution.

### 3.1.4 Summary

There are various kinds of evaluation approaches for virtualization technologies, which can be summarized as observation-based evaluation, responsiveness-based evaluation, and quantitative evaluation. The paper “Virtualization of Linux based computers” ([9]) performs a typical observation-based test to demonstrate the Linux-VServer project is lightweight and mature for virtualizing Linux servers on a Linux operating system host. The responsiveness of the virtualization technologies is tested by recording creation or destruction times as in the evaluation of DVEs ([7]) and VMPlants ([8]). Following that, three quantitative evaluations are described. The comparison of four host virtualization tools ([11]) measures the different scalability parameters for the different machine resources (CPU, memory disk and network) on three scalability metrics (overhead, linearity and isolation). The evaluation of Denali ([16]) presents the performance of primitive operations, the virtual architecture modifications result in enhanced scale, performance, and simplicity, and the system performs at scale. Finally, the test of Xen ([1]) performs the operating benchmarks, the performance of concurrent virtual machines, the performance isolation, and scalability.

In conclusion, the objective of most existing evaluations is to test the functions that a virtualization technology should have, such as transparency, portability, performance isolation, and so on. In addition, quantitative evaluation is the most scientific and convictive method of these three approaches. However, the observation-based evaluation and responsiveness-based evaluation are also indispensable, because the intrinsic attributions can be deduced from them.
3.2 Existing Evaluations of Solaris Zones

Compared with existing evaluations of other virtualization technologies, there are few results published on the performance and scalability of Solaris Zones. Two papers contain information about the performance impact of running zones, which we discuss in the following paragraphs.

A rough comparison of the performance of several workloads (Java application server, time-sharing, networking, and database) running in a zone and the same workloads running without zones are provided in the paper, “Operating System Support for Server Consolidation” ([14]). As can be seen from their results, the performance impact from using zones is small to nonexistent.

In the paper, “Operating System Support for Consolidating Commercial Workloads” ([10]), the authors measured the performance of a variety of workloads when running in a non-global zone, then compared to the same workloads running without zones. The results are shown in Table 3-2. As can be seen, the impact of running an application in a zone is minimal. They also measured the performance of running multiple applications on the system at the same time in different zones. However, no detailed results are provided except a simple conclusion: the performance when using zones was equivalent, and in some cases better, than the performance when running each application on a separate system.

<table>
<thead>
<tr>
<th>Workload</th>
<th>Base</th>
<th>Zone</th>
<th>Diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>38.45</td>
<td>38.29</td>
<td>99.6</td>
</tr>
<tr>
<td>Time-sharing</td>
<td>23332.58</td>
<td>22406.51</td>
<td>96.0</td>
</tr>
<tr>
<td>Networking</td>
<td>283.30</td>
<td>284.24</td>
<td>100.3</td>
</tr>
<tr>
<td>Database</td>
<td>38767.62</td>
<td>37928.70</td>
<td>97.8</td>
</tr>
</tbody>
</table>

Table 3-2: Performance impact of running typical workloads in the base OS vs. in a zone
Chapter 4
Evaluation of Solaris Zones

In this Chapter we propose an evaluation methodology and use it to examine the performance of Solaris Zones. The AIM benchmark is chosen to test the Solaris system. We design three scenarios for the evaluation. The detailed protocol and process will be introduced in the following four sections: test bed, test tool, test protocol and test description.

4.1 Test Bed

In this section, we present the experimental conditions used to evaluate the performance of Solaris Zones. They comprise the system description and specification of zones.

4.1.1 System description

To test the performance of Solaris Zones, we use Sun Fire X2100 x64 servers. They are equipped with AMD Opteron processor (model 175, dual core) running at 2.2GHz with 1MB of cache, 2GB of DDR-400 memory and a hard disk of 80GB. The operation system running on the machine is Solaris and its version we used is SunOS 5.11 snv_44 i86pc.
We have two machines with the same configurations as described above. One is named \textit{adit2-105}; the other is \textit{adit2-106}. \textit{Adit2-106} is configured to run the tests in the presence of a security configuration known as Solaris Trusted Extensions, while \textit{adit2-105} is used for all other tests. Both machines are headless systems controlled remotely from a desktop system that is also running Solaris OS.

### 4.1.2 Specification of zone

The operation of zone creation is executed repeatedly during the evaluation. We apply the same creation process for all the three test scenarios. The zone creation is comprised of four steps: configuration, installation, booting, and login.

- **Configuration**

  We use the \texttt{zonecfg} command to configure a zone by specifying various parameters for the zone’s virtual platform and application environment. To satisfy the need for our evaluation, we write a configuration file as shown below. It is a basic configuration just including the necessary setting: zone name, path, autoboot, file system mounted, shared file system, and virtual network interface.

  ```
  zonecfg -z yuan$i "create ;
  set zonepath=/export/home/yuan$i ;
  set autoboot=true ;
  add fs ; set dir=/usr/local/yuan$i ;
  set special=/opt/local/yuan$i ;
  set type=lofs ; end ;
  add inherit-pkg-dir ; set dir=/opt/sfw/yuan$i ;
  end ;
  add net ; set address=192.168.25.$i ;
  set physical=bge0 ; end "
  ```
Installation

We use the zone administration command `zoneadm` to install software at the package level into the file system hierarchy established for the zone.

`zoneadm -z yuan$i install`

Internal zone configuration

After installation, the zone is in an unconfigured state. The zone does not have an internal configuration for naming services, its locale and time zone have not been set, and various other configuration tasks have not been performed. So we create an `etc/sysidcfg` file and place it inside the zone. The system id file answers the questions of the internal configuration as shown below. The internal configuration is completed at the first time the zone is booted.

```plaintext
system_locale=C
terminal=dtterm
network_interface=primary {hostname=yuan$i
    netmask=255.255.255.0
    protocol_ipv6=no}
security_policy=NONE
name_service=NONE
timezone=Europe/Stockholm
root_password=XXXXXXXXXXXXX
nfs4_domain=dynamic
```

Boot

The `zoneadm` command is used to boot the zone. The zone enters the running state as soon as the following command is completed.

`zoneadm -z yuan$i boot`
4.2 Test Tool

We use the AIM (Advanced Integration Matrix) benchmark tool as our evaluation tool. The main advantage of the AIM benchmark is its applicability for measuring performance influence of different configurations on a single system. The benchmark details are described in the following four subsections: introduction, requirements, configuration and results.

4.2.1 Introduction

The test tool used in this evaluation is called AIM Independent Resource Benchmark – Suite IX Version 1.1, which was developed by Caldera International, Inc. (Caldera) in the USA ([2]). The AIM Independent Resource Benchmark can exercise each component of a UNIX computer system independently. The benchmark uses 60 sub-benchmarks to generate absolute processing rates, in operations per second, for the subsystem, I/O transfers, function calls, and UNIX system calls.

The test results are used to compare different machines on a test-by-test basis or to measure the success or failure of system tuning and configuration changes on a single system.

4.2.2 Requirements

The AIM benchmark can be compiled and run on most POSIX-compliant UNIX systems. The program uses standard system calls and all source code is in ANSI C. To assure its portability in the tested machine, we check the minimum requirements of the AIM benchmark for the tested system. The outcomes were presented in Table 4-1.
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Satisfied or not for SunOS 5.11</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSIX compliant system, or any version thereof</td>
<td>Yes</td>
<td>POSIX compliant system</td>
</tr>
<tr>
<td>make utility</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>ANSI C compiler</td>
<td>Yes</td>
<td>gcc-3-4-3</td>
</tr>
<tr>
<td>Bourne shell</td>
<td>Yes</td>
<td>/bin/sh</td>
</tr>
<tr>
<td>A POSIX-compliant implementation of tar</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>System V or BSD IPC facilities, including shared memory, semaphores, sockets, and pipes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Unloaded system</td>
<td>Yes</td>
<td>New clear OS</td>
</tr>
<tr>
<td>Enough disk space to create necessary temporary files for disk tests</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Table 4-1: Requirement checklist of the AIM benchmark with respect to the Solaris OS (SunOS 5.11 snv_44)

As shown in Table 4-1, all minimum requirements are satisfied for the tested system. Especially, it is a new configured system which is a good platform for obtaining trusted test results. Thus we can benchmark the Solaris system using the AIM Benchmark.

4.2.3 Configuration

To install and start the AIM benchmark, several configurations should be predetermined and installed right in the tested system. The major configurations are listed in Table 4-2.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH</td>
<td>The path to find executable and all associated files.</td>
<td>/usr/bin:/usr/sbin:/usr/ccs/bin:/usr/sfw/bin: .</td>
</tr>
<tr>
<td>s9workfile</td>
<td>The list of the tests that the benchmark should run.</td>
<td>All the tests.</td>
</tr>
<tr>
<td>FILESIZE</td>
<td>The size of temporary files.</td>
<td>5M</td>
</tr>
<tr>
<td>Compiler</td>
<td>Name of compiler executable.</td>
<td>gcc</td>
</tr>
<tr>
<td>Compiler options</td>
<td>The options to optimize code generation.</td>
<td>-O</td>
</tr>
<tr>
<td>Linker options</td>
<td>The linker to the networking libraries.</td>
<td>-lnsl -lssocket</td>
</tr>
<tr>
<td>Bourne shell</td>
<td>The location of Bourne shell executable.</td>
<td>/bin</td>
</tr>
<tr>
<td>Machine name</td>
<td>Identifying name of the tested system.</td>
<td>adit2-105; adit2-106</td>
</tr>
<tr>
<td>Machine configuration</td>
<td>Additional identifying information.</td>
<td>SunOS 5.11 snv_44</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i86pc i386 i86pc</td>
</tr>
<tr>
<td>Time for running each</td>
<td>Amount of time to run each individual test in the benchmark.</td>
<td>10</td>
</tr>
<tr>
<td>Path to disk files</td>
<td>Location for creation of temporary files used during disk tests</td>
<td>/tmp</td>
</tr>
</tbody>
</table>

Table 4-2: Detailed configurations for the AIM benchmark for the Solaris OS

To run the AIM benchmark environment variables must be set correctly to find the executable and all associated files. Because bash is used as the shell in this evaluation, the file `~/.bashrc` could be created to set the variable of `$PATH`. The value in this case is `/usr/bin:/usr/sbin:/usr/ccs/bin:/usr/sfw/bin: .`, in which `/usr/ccs/bin:` specifies the path to command `make` and `/usr/sfw/bin:` points to gcc compiler.

Before installing the AIM benchmark, the benchmark’s workfile, `s9workfile`, must be configured so that the test environment is specified properly. All 60 sub-benchmarks are listed in the `s9workfile` and the benchmark reads the file each time when it is run. Thus, altering the workfile can specify which tests would be run according to the test environment. An unmodified version of this
file is left to cause the benchmark to run all 60 sub-benchmarks in this evaluation. Additionally, FILESIZE, a parameter in the s9workfile, defines the size of temporary files for disk tests, which we set to 5MB.

After specifying the file s9workfile, the s9setup script can be run to set up the benchmark. It needs four variables provided manually. The first is the name of the C compiler. In the tested system, gcc is decided to be employed as the C compiler. Secondly, compiler option -O is used to optimize code produced for gcc. Then, the linker options for the InterProcess Communication tests should be -lnsl -lsocket. Finally, the location of the Bourne shell executable on this system is /bin. After responding to all the script’s queries, command make should be entered to generate executables.

Among these executables, the singleuser program is used to run the benchmark. To start the benchmark, it requires the following information: machine name, machine configuration, numbers of seconds to run each test [2 to 1000] and location for creation of temporary files used during disk tests. For this evaluation, the values are **adit2-105** or **adit2-106**, **SunOS 5.11 snv_44 i86pc i386 i86pc**, **10** and **/tmp** respectively.

**4.2.4 Results**

The singleuser program generates a tab delimited spreadsheet output file, suite9.ss, which contains the benchmark results. The contents include the name and configuration of the machine, the date of the benchmark run, the name of the benchmark, the disk iteration count, and a tab separated line for each test run with name of test, timing, and units.

Furthermore, the 60 sub-benchmarks of the AIM benchmark can be summarized in the following seven parameters:
- **Arithmetic**
  It contains 15 sub-benchmarks for measuring operations for five types of data (Short, Int, Long, Float and Double). The tested operations are addition, division and multiplication.

- **Memory and Process Management**
  There are 4 sub-benchmarks for evaluating Memory and Process Management. They evaluate the System Memory Allocations, Paging, Program Loads, and Task Creations.

- **Disk/Filesystem I/O**
  The sub-benchmarks for Disk/Filesystem I/O consist of various read and write tests, such as Sequential Disk Reads, Random Disk Writes, and Sync Disk Copies. Directory Searches, File Creations and Link/Unlink Pairs tests are also the important components for evaluating Disk/Filesystem I/O.

- **Function Calls**
  It contains 4 sub-benchmarks to evaluate Function Calls with different number of arguments, such as no arguments, 1 argument, 2 arguments, and 15 arguments.

- **InterProcess Communication**
  We have 7 sub-benchmarks to collect the communication performance from following aspects: TCP/IP, UDP/IP, FIFOs, PIPEs, Shared RAM, Stream Pipe, and DataGram Pipe.

- **Library/System**
  There are 13 sub-benchmarks for gathering the performance of Library/System. They are comprised of Jump Test, Sort Operations, Signal Traps, and several function tests for numeral, trigonometric, string, and shell.

- **Algorithmic Tests**
  There are five algorithmic tests, which are for Newton-Raphson, 3D Projection, Linear System, Taylor Series, and Integer Sieves.
4.3 Test Protocol

Our test protocol consists of two parts: scalability and performance. For every part, we describe our method to evaluate Solaris Zones. These methods are then turned into practical tests in the next section to characterize the performance of the Solaris system.

4.3.1 Scalability

In this part, we examine the Solaris system’s ability of installing zones by recording the creation time. It is an important characteristic of a virtualization technology ([10]) and also a part of usability for operating system. We consider the following metric: the time to create the nth zone, all n-1 zones being in running state. Let $T_{create}$ be the time to create the nth zone, we have $T_{create} = T_{configure} + T_{install} + T_{boot}$, where $T_{configure}$, $T_{install}$, $T_{boot}$ mean the configuration time, installation time, and booting time for the nth zone respectively. This metric can be affected by different services required during the configuration sequence.

4.3.2 Performance

In this part, we evaluate the performance of Solaris Zones on the Solaris OS in three scenarios. In the first scenario we measure - as a baseline - the performance of Solaris Zones on a 2-CPU core machine in the standard configuration that is distributed as part of the Solaris OS. In the second scenario we investigate the influence of the number of CPU cores, by using the same tests on just a single CPU core. In the third scenario we evaluate the performance in the presence of a security configuration known as Solaris Trusted Extensions. The detailed protocols are described in the following subsections.
4.3.2.1 Baseline performance of Solaris Zones

To evaluate the performance impact of the Solaris operating system, we should record the performance of Solaris in its standard configuration. It is also the baseline for the next two parts. We examine the performance of Solaris Zones as following: performance of the global zone, performance of a non-global zone, and performance of increasing numbers of concurrently running non-global zones.

- **Performance of the global zone**
  To evaluate changes in the performance of global zone, we run the benchmark in global zone as the number of running zones increases. There should be $n+1$ results of the benchmark after creating $n$ non-global zones: one for only the global zone existed ($R_0$), while $n$ for $n$ non-global zones created respectively ($R_1, R_2 ..., R_n$).

- **Performance of a non-global zone**
  To examine the performance of a non-global zone as the number of zones grows, we run the benchmark in the first non-global zone. In this situation, only a single zone actually runs the benchmark and the other zones including the global zone are free of application. Let $R'_1$ be the result when only global zone and the first non-global zone exist, we have $R'_{n}$ for representing the result after creating $n$ non-global zones.

- **Performance of running benchmarks concurrently in different zones**
  To evaluate the performance of running multiple instances of the benchmark in different zones concurrently, we first measure the benchmark on the first non-global zone. As the number of zones scales up, we run the benchmark concurrently in each running non-global zone. In this situation, only the global zone is idle and all of non-global zones are executing the
same benchmark at the same time after they are in running state. We let $R_{in}$ be result of the benchmark running on the $i^{th}$ non-global zone when $n$ non-global zone exist. Thus the total value of running the benchmark when $n$ non-global zones exist in the system is:

$$S_n := \sum_{i=1}^{n} R_{in}$$

### 4.3.2.2 Performance of running on a single CPU core

This is a comparative test with our baseline test described above. We run the same tests on just a single CPU core, so that the influence of the number of CPU cores can be presented obviously. To be simple and avoid redundancy, only the performance of the global zone is collected in this case.

### 4.3.2.3 Performance of security configuration enabled

To evaluate the performance impact of the Solaris system with security configuration enabled, we design two evaluation protocols. The first one is used to evaluate the performance of the global zone as the number of zones increases, while the other is used for aggregate performance when running multiple instances of the benchmark concurrently in different zones.

- **Performance of the global zone**

  This protocol is used to find changes in performance of the global zone with security configuration enabled and disabled. We evaluate performance of the global zone in the same way as the one we used for collecting the baseline performance of Solaris Zones. Thus, the benchmark is applied in the global zone repeatedly as the number of zones increases. We denote $L_n$ to be the result of the benchmark when $n$ non-global zones are running.
Performance of running benchmarks concurrently in different zones

We examine the performance of running multiple instances of the benchmark concurrently in the same way as the one described in the baseline performance of Solaris Zones. As the number of zones scales up, the benchmark is running concurrently in each running non-global zone. We denote \( L_{in} \) to be result of the benchmark running in the \( i^{th} \) non-global zone when \( n \) non-global zone exist. The aggregate value of results when there are \( n \) non-global zones in the system is: \( S_n := \sum_{i=1}^{n} L_{in} \).

4.4 Test Description

We use two test machines, named \textit{adit2-105} and \textit{adit2-106}, whose hardware configuration and OS version are completely identical ([§4.1.1]). On \textit{adit2-105} we run the tests of the first two scenarios we introduced above, while machine \textit{adit2-106} is configured to run the tests with security configuration enabled. In addition, we have created some scripts to automate the test process and all the scripts are running in the global zone. They are included in Appendix.

4.4.1 Tests on \textit{adit2-105}

We execute four tests on machine \textit{adit2-105}. The first test is used to gather creation time of zones and baseline performance of the global zone, while the second one records the baseline performance of a non-global zone. Then the third one traces baseline performance of running multiple instances of benchmark concurrently in different zones. Finally, the performance of running on one CPU core is reflected in the fourth test. To make sure an unloaded and clear system is used for each test, we delete all the non-global zones and reboot the machine before every test.
Creation time and performance of the global zone

To collect the time of zone’s creation and performance of the global zone as the number of zones grows, we run the benchmark every time when a new zone is booted successfully. Because every result of the benchmark contains the starting time and ending time of the execution, the creation time of the nth zone \( T_{create(n)} \) can be calculated as the following: \( T_{create(n)} = T_{start(n)} - T_{end(n-1)} \). \( T_{start(n)} \) is the starting time of the benchmark which is executed after creating the \( n^{th} \) zone, while \( T_{end(n-1)} \) is the ending time of the benchmark that is performed after creating the \( (n-1)^{th} \) zone. Figure 4-1 shows the flow of this test. A script is created to run the test without the need for human interaction. It is included in Appendix a).
Performance of a non-global zone

In this test, we focus on the performance of a non-global zone. The flow chart is similar to Figure 4-1, with one exception: we run the benchmark in the first non-global zone for the second rounded rectangle. Because the job of zone creation must be executed in the global zone, the script cannot be run in the non-global zone. Thus, we use the command `zlogin` to force the
benchmark running under the non-global zone. The detailed script is included in Appendix b).

- **Performance of running benchmarks concurrently in different zones**

To evaluate the performance of running multiple instances of the benchmark concurrently, we execute the benchmark in different zones at the same time after creating a new zone. We also run the script in the global zone and use the command `zlogin` to execute the benchmark under each non-global zone. The difference of the command from above is that we run the command `zlogin` in the background, so that the system can execute the next command without waiting till the current one has finished. In this case, we can automate multiple instances of the benchmark at the same time in different zones. The relative flow chart is displayed in Figure 4-2 and the script is included in Appendix c).
Figure 4-2: Evaluation flow chart for the performance of running benchmarks concurrently in different zones

- **Performance of running on only one CPU core**
  
  Once the tested machine is booted, it has two CPU cores online. The basic performance gathered above is under the initial situation of two CPU cores being online. In this test, we cut the computing resources into half by halting one CPU core. To evaluate changes in the performance of this scenario, we run the same test as the one for baseline performance of the global zone.
4.4.2 Tests on adit2-106

The security mechanism we applied on the Solaris OS is made available through Solaris Trusted Extensions. It enables the Solaris OS to support both traditional Discretionary Access Control (DAC) policies based on file and device ownership, as well as label-based Mandatory Access Control (MAC) policies. Trusted Extensions associates labels with Solaris Zones. Each zone is assigned a unique sensitivity label and can be customized with its own set of file systems and network resources. All the zones are centrally administered from the global zone. MAC policy enforcement is automatic in labeled zones and applies to all their processes.

All systems that are configured with Trusted Extensions have labels. Labels are used to implement and control access on a computer. With Trusted Extensions, both discretionary access control (DAC) checks and MAC checks must pass before access is allowed to an object. In the Solaris OS, DAC is based on permission bits and access control lists (ACLs). MAC compares the label of a process that is running an application with the label or the label range of any object that the process tries to access.

Labels consist of a hierarchical component called classification and a non-hierarchical component called compartment. The classification portion of a label indicates a relative level of protection. The compartment portion of a label is optional. Compartment words in a label can be used to represent different kinds of groupings, such as work groups, departments, divisions, or geographical areas.

Trusted Extensions supports Common IP Security Option (CIPSO) labels ([13]). Each label has a classification field that allows 256 values, and a 256-bit compartments field. We cannot use 0 (zero) for a classification, so we can
define a total of 255 classifications. For CIPS0 labels, 240 compartment bits are available, for a total of $2^{240}$ compartment combinations. The components are illustrated in Figure 4-3.

![CIPS0 Label Definition](image)

**Figure 4-3: CIPS0 Label Definition**

Labels are specified in a *label_encodings* file, which is a flat text file in the `/etc/security/tsol` directory. We can define different labels for our system through editing the *label_encodings* file. Additionally, there are two types of session label: single-level and multilevel sessions. In a single-label session, we can access only those objects that are equal to our session label or are dominated by the label. In a multilevel session, we can access information at sensitivity labels that are equal to or lower than our session clearance. In our evaluation, we specify to operate at single-level session.

After finishing the preparation of the Solaris OS, we installed Solaris Trusted Extensions by loading the Trusted Extensions software packages on `adit2-106`. Then we created a *label_encodings* file containing 120 different labels. As shown in Table 4-3, we define four classifications, named Public, Internal-use-only, Need-to-know and Registered respectively. For the Need-to-know classification, we specify 117 labels with different compartment values.
After rebooting the system, the `label_encodings` file is taking effect. We configure 120 zones and launch the Solaris Management Console. By using the Console, we associate the appropriate label with a zone name.

Two tests are run on **edit2-106**: one for the performance of the global zone and the other for the performance of running multiple instances of the benchmark concurrently in different zones. To compare with the baseline scenario, the protocol of the two tests is identical to the corresponding one in the baseline scenario. Thus, the flow chart and scripts are also identical to the one in the global zone and the one for concurrent performance in the baseline scenario.
Chapter 5
Results and Analysis

In this chapter, we present and analyze the evaluation results. The results and analysis are organized in two major aspects: scalability and performance. In the first section, we evaluate how well the operating system scales with respect to an increasing number of zones. Then, the following section discusses the performance of Solaris Zones in three scenarios.

5.1 Scalability

In this section, we evaluate the scalability of Solaris Zones by investigating zone creation time. Creation times are influenced as the number of zones increases. A closer look at the profile of creation times as a growing number of zones shown in Figure 5-1.
The creation time is the sum of configure time, installation time and booting time per zone. Figure 5-1 shows the creation times from the 1st non-global zone to the 126th non-global zone. In this case, all the other n-1 non-global zones are in running state, when the nth zone was created. For small numbers of zones, there is no apparent rise in creation times. However, a rapid increase can be observed with minor rebounds starting with the 110th zone. The creation time of the 110th zone is 68s, while the time of the 126th zone sharply increases to 471s.
A closer look at the creation times from the 1st to the 111st non-global zone is shown in Figure 5-2. Each creation time of the 1st to the 109th non-global zones is under 15 minutes. Particularly, it costs about 5 minutes for successfully creating a zone when no more than 86 non-global zones exist in the system. From the 86th to the 109th zone, the creation time increases to approximate 13 minutes per zone.

In conclusion, the creation time of the 1st to the 109th non-global zones is no more than 15 minutes for each. However, from the 110th non-global zone, the creation times jump quickly. It costs one hour or even more to create one zone. It is no longer an acceptable performance for installing zones. The amount of time that is considered acceptable for zone configure/install/boot is not a universally agreed upon value, but user dependent. We considered a value of up to 15 minutes acceptable. Thus we choose 109 non-global zones as the maximum number of zones in our performance evaluation part.
5.2 Performance

In this section, we present and discuss our benchmark results. The benchmark evaluation is performed in three scenarios. To compare the performance of different scenarios, we analyze the results from the following two aspects: the performance of running the benchmark in one zone and the performance of running multiple instances of benchmark concurrently in different zones.

5.2.1 Performance of running the benchmark in one zone

In this part, we only present and compare the results of running a single instance of the benchmark in the system. In our evaluation, four sub-evaluations meet this requirement. Firstly, we examine the global zone and a non-global zone respectively in the baseline scenario. The baseline scenario can be described as a two-CPU core Solaris system using the Solaris Zones mechanism without any security configuration. Secondly, we record the performance of the global zone in the scenario of only one CPU core online. Finally, we collect the performance of the global zone after enabling the security configuration.

The performance of the global zone and non-global zone is evaluated as the number of zones increases. Because our system can run up to 109 non-global zones according to the creation time mentioned in section 5.1, we benchmark the performance from a system containing one zone (only the global zone exists in the system) up to a system containing 110 zones (the global zone plus 109 non-global zones).
<table>
<thead>
<tr>
<th>Categories</th>
<th>Sub-benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory and Process Management</td>
<td>System Memory Allocations, System Memory Allocations &amp; Pages, Program Loads, Task Creations</td>
</tr>
<tr>
<td>Function Calls</td>
<td>Function Calls (no argument), Function Calls (1 argument), Function Calls (2 arguments), Function Calls (15 arguments)</td>
</tr>
</tbody>
</table>

Table 5-1: The Classification of the AIM benchmark

As shown in Table 5-1, the AIM benchmark consists of 60 sub-benchmarks related to Arithmetic, Disk/Filesystem I/O, Memory, InterProcess Communication and so on. We present one sub-benchmark as a representative example. Figure 5-3 shows the Integer Additions performance of the global zone (dashed line) and a non-global zone (dash-dotted line) as the number of zones increases in the baseline scenario. The straight dashed line is the corresponding linear regression line for the global zone, while the solid line presents the corresponding linear regression line for the non-global zone. To
analyze a great number of benchmark results, we define the following variables. As shown in Figure 5-3, “Δ” is the difference of the function value of the linear regression line for the 109th zone and its function value for the 1st zone. “d” indicates the average distance between the non-global zone trendline and the global zone trendline.

![Figure 5-3: Integer Additions performance of the global and non-global zone as the number of zones increases in the baseline scenario](image)

Let \( y = ax + b \) be the equation of trendline, we have \( \Delta\% = \Delta / b \) showing the performance trend and changes of the zone itself as the number of zones increases. A positive value of \( \Delta\% \) indicates an ascending trend, while a negative value indicates a falling trend. Also, let \( y_0 = a_0x + b_0 \) be the trendline equation of the global zone performance in the baseline scenario, we have \( d\% = d / b_0 \) to calculate the relative differences between the baseline global zone and other situations. A positive value of \( d\% \) indicates the performance of tested zone is better than the baseline global zone, while a negative one indicates a
worse performance. Table 5-2 shows the brief description of the variables we defined above.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Δ</td>
<td>The difference of the function value of the linear regression line for the 109th zone and its function value for the 1st zone.</td>
</tr>
<tr>
<td>Δ%</td>
<td>System performance impact as the number of zones increases</td>
</tr>
<tr>
<td>d</td>
<td>The average distance between the non-global zone trendline and the global zone trendline.</td>
</tr>
<tr>
<td>d%</td>
<td>System performance difference compared with the baseline metrics</td>
</tr>
</tbody>
</table>

Table 5-2: The description of the relevant variables in the performance evaluation

In the following part, we analyze the performance of the global zone and non-global zones from seven aspects. The parameters for the seven aspects consist of **Arithmetic**, **Disk/Filesytem I/O**, **Memory Management**, **Function Call**, **InterProcess Communication**, **Library/System**, and **Algorithmic Tests**.

- **Arithmetic**

In the AIM benchmark we have 15 sub-benchmarks for **Arithmetic**. They record the results of three arithmetical operations (addition, divide, and multiply) acted on five data types (short, int, long, float, and double). Table 5-3 presents the operation performance of ‘Int’ and ‘Long’, while Table 5-4 shows the performance of ‘Float’ and ‘Double’.
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Category</th>
<th>Trendline Equation</th>
<th>Int Additions</th>
<th>Int Divides</th>
<th>Int Multiplies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Global</td>
<td>$y = -755.25x + 4299360$</td>
<td>$y = -0.9693x + 53635$</td>
<td>$y = -22.325x + 1017087$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Δ %</td>
<td>-1.91%</td>
<td>-0.20%</td>
<td>-0.24%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Non-global</td>
<td>$y = -105.76x + 4207347$</td>
<td>$y = -1.3189x + 53571$</td>
<td>$y = -9.6558x + 1015402$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Δ %</td>
<td>-0.27%</td>
<td>-0.27%</td>
<td>-0.10%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d %</td>
<td>-1.31%</td>
<td>-0.16%</td>
<td>-0.10%</td>
<td></td>
</tr>
<tr>
<td>1 CPU core</td>
<td>Global</td>
<td>$y = -900.79x + 3370946$</td>
<td>$y = -7.3525x + 50703$</td>
<td>$y = 9.4675x + 1000140$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Δ %</td>
<td>-2.178%</td>
<td>-1.58%</td>
<td>0.10%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d %</td>
<td>-1.94%</td>
<td>-6.12%</td>
<td>-1.49%</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>Global</td>
<td>$y = -1274.9x + 4225011$</td>
<td>$y = -9.4028x + 53577$</td>
<td>$y = -285.3x + 1014382$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Δ %</td>
<td>-3.68%</td>
<td>-2.34%</td>
<td>-2.11%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d %</td>
<td>-0.60%</td>
<td>-1.08%</td>
<td>-1.82%</td>
<td></td>
</tr>
<tr>
<td>Base-line</td>
<td>Global</td>
<td>$y = 1147.5x + 4045252$</td>
<td>$y = -0.8203x + 53619$</td>
<td>$y = -16.091x + 1016445$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Δ %</td>
<td>3.09%</td>
<td>-0.17%</td>
<td>-0.17%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Non-global</td>
<td>$y = 970.99x + 4052174$</td>
<td>$y = -1.74x + 53574$</td>
<td>$y = -39.949x + 1015785$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Δ %</td>
<td>2.61%</td>
<td>-0.35%</td>
<td>-0.43%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d %</td>
<td>-0.07%</td>
<td>-0.18%</td>
<td>-1.94%</td>
<td></td>
</tr>
<tr>
<td>1 CPU core</td>
<td>Global</td>
<td>$y = -726.74x + 3272180$</td>
<td>$y = 4.7818x + 51885$</td>
<td>$y = 237.76x + 1008125$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Δ %</td>
<td>-2.42%</td>
<td>1.00%</td>
<td>-2.57%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d %</td>
<td>-19.68%</td>
<td>-2.66%</td>
<td>-2.02%</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>Global</td>
<td>$y = -1155.9x + 4027305$</td>
<td>$y = -8.8223x + 53425$</td>
<td>$y = -267.9x + 1013785$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Δ %</td>
<td>-3.77%</td>
<td>-1.87%</td>
<td>-2.67%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d %</td>
<td>-0.81%</td>
<td>-1.54%</td>
<td>-1.85%</td>
<td></td>
</tr>
</tbody>
</table>

Table 5-3: The performance of Int and Long arithmetical operations in three different scenarios (Thousand Ops per Second)
<table>
<thead>
<tr>
<th>Baseline</th>
<th>Float Additions</th>
<th>Float Divides</th>
<th>Float Multiplies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>$y=91.643x+1034875$</td>
<td>$y=-4.0738x+210268$</td>
<td>$y=-22.079x+1103692$</td>
</tr>
<tr>
<td>Δ %</td>
<td>0.97%</td>
<td>-0.75%</td>
<td>-0.22%</td>
</tr>
<tr>
<td>Non-global</td>
<td>$y=63.008x+1034270$</td>
<td>$y=-4.4327x+209976$</td>
<td>$y=-25.066x+1102145$</td>
</tr>
<tr>
<td>Δ %</td>
<td>0.66%</td>
<td>-0.23%</td>
<td>-0.25%</td>
</tr>
<tr>
<td>d %</td>
<td>-0.21%</td>
<td>-0.15%</td>
<td>-0.16%</td>
</tr>
<tr>
<td>Security</td>
<td>$y=-409.13x+1010739$</td>
<td>$y=-65.63x+203265$</td>
<td>$y=-354.9x+1004330$</td>
</tr>
<tr>
<td>Δ %</td>
<td>-4.28%</td>
<td>-2.80%</td>
<td>-3.51%</td>
</tr>
<tr>
<td>d %</td>
<td>-2.35%</td>
<td>-2.97%</td>
<td>-4.18%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Double Additions</th>
<th>Double Divides</th>
<th>Double Multiplies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>$y=112.8x+1019148$</td>
<td>$y=-4.4875x+210328$</td>
<td>$y=-19.874x+1103413$</td>
</tr>
<tr>
<td>Δ %</td>
<td>1.21%</td>
<td>-0.23%</td>
<td>-0.20%</td>
</tr>
<tr>
<td>Non-global</td>
<td>$y=126.67x+1013268$</td>
<td>$y=-5.2534x+210091$</td>
<td>$y=-10.947x+1101832$</td>
</tr>
<tr>
<td>Δ %</td>
<td>1.36%</td>
<td>-0.27%</td>
<td>-0.11%</td>
</tr>
<tr>
<td>d %</td>
<td>-0.50%</td>
<td>-0.13%</td>
<td>-0.10%</td>
</tr>
<tr>
<td>Security</td>
<td>$y=-98.423x+808146$</td>
<td>$y=-47.729x+208510$</td>
<td>$y=161.93x+1059107$</td>
</tr>
<tr>
<td>Δ %</td>
<td>-1.33%</td>
<td>-2.50%</td>
<td>1.67%</td>
</tr>
<tr>
<td>d %</td>
<td>-21.84%</td>
<td>-2.00%</td>
<td>-3.11%</td>
</tr>
</tbody>
</table>

**Table 5-4: The performance of Float and Double arithmetical operations in three different scenarios (Thousand Ops per Second)**

We evaluate these results from two important aspects. The first aspect is the performance impact as the number of zones increases. The second aspect is the performance difference between our three scenarios compared to the OS’ baseline performance in the global zone. As shown in Table 5-3 and Table 5-4, the performance changes for these four data types are quite similar. The performance of arithmetic operations shows a falling trend as the number of zones increases. However, Δ% of each scenario is less than 2% of performance decrease for most sub-benchmarks. Thus the performance impact is quite small as the number of zones increases.
The performance difference in our situations stands out. The performance for Additions in the one-CPU core scenario is about -20% compared to the performance of the global zone in the baseline scenario.

➢ Memory and Process Management

<table>
<thead>
<tr>
<th></th>
<th>Memory Allocations</th>
<th>Memory Allocations &amp; Pages</th>
<th>Program Loads</th>
<th>Task Creations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global Trendline</td>
<td>$y = -729.53x + 6852470$</td>
<td>$y = -55.827x + 114285$</td>
<td>$y = -0.7036x + 637.25$</td>
<td>$y = -0.845x + 1442.7$</td>
</tr>
<tr>
<td>Δ %</td>
<td>-9.33%</td>
<td>-5.32%</td>
<td>-12.03%</td>
<td>-6.38%</td>
</tr>
<tr>
<td>d %</td>
<td>0.14%</td>
<td>-0.78%</td>
<td>-3.58%</td>
<td>-1.51%</td>
</tr>
<tr>
<td><strong>Non-global</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trendline</td>
<td>$y = -948.01x + 865667$</td>
<td>$y = -58.407x + 113533$</td>
<td>$y = -0.7272x + 615.75$</td>
<td>$y = -0.7402x + 1415.2$</td>
</tr>
<tr>
<td>Δ %</td>
<td>-11.94%</td>
<td>-5.61%</td>
<td>-12.87%</td>
<td>-5.70%</td>
</tr>
<tr>
<td>d %</td>
<td>0.14%</td>
<td>-0.78%</td>
<td>-3.58%</td>
<td>-1.51%</td>
</tr>
<tr>
<td><strong>1 CPU core</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global Trendline</td>
<td>$y = -515.82x + 674268$</td>
<td>$y = -92.274x + 94221$</td>
<td>$y = -2.0161x + 644.55$</td>
<td>$y = -1.1609x + 1490.8$</td>
</tr>
<tr>
<td>Δ %</td>
<td>-8.34%</td>
<td>-10.67%</td>
<td>-34.09%</td>
<td>-8.49%</td>
</tr>
<tr>
<td>d %</td>
<td>-19.53%</td>
<td>-19.31%</td>
<td>-10.18%</td>
<td>2.13%</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global Trendline</td>
<td>$y = -375.39x + 573436$</td>
<td>$y = -85.447x + 104263$</td>
<td>$y = -1.3602x + 622.66$</td>
<td>$y = -2.2677x + 1418.2$</td>
</tr>
<tr>
<td>Δ %</td>
<td>-7.22%</td>
<td>-9.68%</td>
<td>-25.44%</td>
<td>-14.45%</td>
</tr>
<tr>
<td>d %</td>
<td>-35.22%</td>
<td>13.66%</td>
<td>-16.46%</td>
<td>-12.29%</td>
</tr>
</tbody>
</table>

Table 5-5: The performance of Memory and Process Management in three different scenarios (Ops per Second)

As shown in Table 5-5, the performance impact of Memory and Process Management is not small as the number of zones grows up. The average Δ% is 10% for each sub-benchmark. Particularly, the performance of Program Loads drops rapidly as the number of zones increases. Its Δ% is up to 34.09% in the one-CPU core scenario and 25.44% in the security-enabled scenario.

The performance difference is apparent in the one-CPU core and security-enabled scenarios. The decline in the one-processor scenario is 19.53%, 19.31% and 10.18% for Memory Allocations, Memory Allocations & Pages, and Program Loads respectively. The decline in the security-enabled scenario is about 13%. The decline of Memory Allocations is up to 35.22%.

56
### Disk/Filesystem I/O

<table>
<thead>
<tr>
<th></th>
<th>Random Reads</th>
<th>Disk Reads</th>
<th>Sequence Disk Reads</th>
<th>Random Writes</th>
<th>Disk Writes</th>
<th>Sequence Disk Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trendline</td>
<td>$y = -69.784x + 212841$</td>
<td>$y = 31.551x + 685209$</td>
<td>$y = -93.981x + 222885$</td>
<td>$y = -223.34x + 379232$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\Delta %$</td>
<td>-3.51%</td>
<td>0.50%</td>
<td>-4.60%</td>
<td>-6.42%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$d %$</td>
<td>-0.40%</td>
<td>-0.26%</td>
<td>-0.51%</td>
<td>-0.73%</td>
<td></td>
</tr>
<tr>
<td><strong>Non-global</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trendline</td>
<td>$y = -86.338x + 212897$</td>
<td>$y = 35.948x + 683185$</td>
<td>$y = -96.613x + 221896$</td>
<td>$y = -232.7x + 376980$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\Delta %$</td>
<td>-4.42%</td>
<td>0.57%</td>
<td>-4.75%</td>
<td>-6.73%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$d %$</td>
<td>-6.37%</td>
<td>-2.16%</td>
<td>-3.00%</td>
<td>-3.21%</td>
<td></td>
</tr>
<tr>
<td><strong>1 CPU core</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Global</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trendline</td>
<td>$y = -74.416x + 199544$</td>
<td>$y = 42.912x + 669760$</td>
<td>$y = -116.87x + 217450$</td>
<td>$y = -160.89x + 363614$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\Delta %$</td>
<td>-4.06%</td>
<td>0.70%</td>
<td>-5.86%</td>
<td>-4.82%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$d %$</td>
<td>-3.67%</td>
<td>-2.16%</td>
<td>-3.00%</td>
<td>-3.21%</td>
<td></td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Global</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trendline</td>
<td>$y = -79.54x + 139277$</td>
<td>$y = -190.57x + 488736$</td>
<td>$y = -123.54x + 147199$</td>
<td>$y = -261.47x + 281675$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\Delta %$</td>
<td>-4.14%</td>
<td>-3.56%</td>
<td>-6.11%</td>
<td>-5.98%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$d %$</td>
<td>-32.73%</td>
<td>-30.65%</td>
<td>-34.31%</td>
<td>-24.87%</td>
<td></td>
</tr>
</tbody>
</table>

### Disk Copy, File Creations and Closes, Directory Searches, Directory Operations

<table>
<thead>
<tr>
<th></th>
<th>Disk Copy</th>
<th>File Creations and Closes</th>
<th>Directory Searches</th>
<th>Directory Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trendline</td>
<td>$y = -77.201x + 231954$</td>
<td>$y = -65.568x + 265625$</td>
<td>$y = -7.1905x + 113979$</td>
</tr>
<tr>
<td></td>
<td>$\Delta %$</td>
<td>-3.63%</td>
<td>-2.69%</td>
<td>-0.69%</td>
</tr>
<tr>
<td></td>
<td>$d %$</td>
<td>-0.45%</td>
<td>-1.76</td>
<td>-2.27%</td>
</tr>
<tr>
<td><strong>Non-global</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trendline</td>
<td>$y = -87.903x + 231506$</td>
<td>$y = -56.035x + 260419$</td>
<td>$y = -7.5715x + 111407$</td>
</tr>
<tr>
<td></td>
<td>$\Delta %$</td>
<td>-4.14%</td>
<td>-2.35%</td>
<td>-0.74%</td>
</tr>
<tr>
<td></td>
<td>$d %$</td>
<td>-5.90%</td>
<td>-24.67%</td>
<td>-6.67%</td>
</tr>
<tr>
<td><strong>1 CPU core</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Global</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trendline</td>
<td>$y = -81.367x + 218495$</td>
<td>$y = -46.906x + 199080$</td>
<td>$y = -16.659x + 106898$</td>
</tr>
<tr>
<td></td>
<td>$\Delta %$</td>
<td>-4.06%</td>
<td>-2.57%</td>
<td>-1.70%</td>
</tr>
<tr>
<td></td>
<td>$d %$</td>
<td>-5.67%</td>
<td>-3.13%</td>
<td>5.78%</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Global</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trendline</td>
<td>$y = -111.46x + 158545$</td>
<td>$y = -71.473x + 155518$</td>
<td>$y = -87.317x + 68699$</td>
</tr>
<tr>
<td></td>
<td>$\Delta %$</td>
<td>-5.67%</td>
<td>-3.13%</td>
<td>5.78%</td>
</tr>
<tr>
<td></td>
<td>$d %$</td>
<td>-30.90%</td>
<td>-43.16%</td>
<td>-27.50%</td>
</tr>
</tbody>
</table>

Table 5-6: The performance of Disk/Filesystem I/O in three different scenarios (Kbytes per Second and Ops per Second)
As shown in Table 5-6, the performance impact of Disk/Filesystem I/O is slightly larger than in the arithmetic benchmarks, but it is less than 4% for most sub-benchmarks. The apparent performance difference appears on most sub-benchmarks in the security-enabled scenario, as well as in one sub-benchmark in the one-CPU core scenario. The performance decline in the security enable scenario is about 30% on average. And the decline of File Creations and Closes in the one-CPU core scenario is 24.67%.

**Function Calls**

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Global</th>
<th>Non-global</th>
<th>1 CPU core</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td>trendline</td>
<td>y = -3038.1x + 146295031</td>
<td>y = -1024x + 293307596</td>
<td>y = -11306x + 302783802</td>
</tr>
<tr>
<td>d %</td>
<td>-0.19%</td>
<td>-0.33%</td>
<td>-0.36%</td>
<td>-0.20%</td>
</tr>
<tr>
<td><strong>1 CPU core</strong></td>
<td>trendline</td>
<td>y = -2544.4x + 146029979</td>
<td>y = -8945.2x + 293138292</td>
<td>y = -9890.2x + 301970278</td>
</tr>
<tr>
<td>d %</td>
<td>-0.16%</td>
<td>-0.21%</td>
<td>-0.24%</td>
<td>-0.14%</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>trendline</td>
<td>y = 23488x + 284060215</td>
<td>y = -34677x + 286275949</td>
<td>y = -21571x + 94850294</td>
</tr>
<tr>
<td>d %</td>
<td>-1.48%</td>
<td>-2.69%</td>
<td>-5.88%</td>
<td>-2.12%</td>
</tr>
</tbody>
</table>

Table 5-7: The performance of Function Calls in three different scenarios (Ops per Second)

Table 5-7 shows that Δ% and d % are less than 2% of performance decrease for most sub-benchmarks of Function Calls in all the three scenarios. Thus, the performance impact as the number of zones increases and difference in various scenarios are quite small for Function Calls.
InterProcess Communication

<table>
<thead>
<tr>
<th>Base-line</th>
<th>TCP/IP Messages</th>
<th>UDP/IP Messages</th>
<th>FIFO Messages</th>
<th>Pipe Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>trendline y = -27.893x + 267893</td>
<td>y = 15.71x + 222503</td>
<td>y = -30.292x + 259083</td>
<td>y = 48.345x + 662624</td>
</tr>
<tr>
<td>Δ %</td>
<td>-1.13%</td>
<td>0.77%</td>
<td>-1.27%</td>
<td>0.80%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-global</th>
<th>TCP/IP Messages</th>
<th>UDP/IP Messages</th>
<th>FIFO Messages</th>
<th>Pipe Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>trendline y = -21.383x + 265697</td>
<td>y = -11.13x + 223797</td>
<td>y = -67.88x + 261309</td>
<td>y = -30.317x + 665432</td>
</tr>
<tr>
<td>Δ %</td>
<td>-0.88%</td>
<td>-0.54%</td>
<td>-2.83%</td>
<td>-0.50%</td>
</tr>
<tr>
<td>d %</td>
<td>-0.69%</td>
<td>-0.08%</td>
<td>0.06%</td>
<td>-0.23%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1 CPU core</th>
<th>TCP/IP Messages</th>
<th>UDP/IP Messages</th>
<th>FIFO Messages</th>
<th>Pipe Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>trendline y = -99.767x + 265623</td>
<td>y = -32.729x + 221338</td>
<td>y = -69.494x + 244900</td>
<td>y = 85.186x + 629006</td>
</tr>
<tr>
<td>Δ %</td>
<td>-4.10%</td>
<td>-1.61%</td>
<td>-3.09%</td>
<td>1.48%</td>
</tr>
<tr>
<td>d %</td>
<td>-2.32%</td>
<td>-1.72%</td>
<td>-6.31%</td>
<td>-4.77%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security</th>
<th>TCP/IP Messages</th>
<th>UDP/IP Messages</th>
<th>FIFO Messages</th>
<th>Pipe Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>trendline y = -38.249x + 112937</td>
<td>y = -85.044x + 134455</td>
<td>y = -115.7x + 166170</td>
<td>y = -178.71x + 366442</td>
</tr>
<tr>
<td>Δ %</td>
<td>-1.89%</td>
<td>-3.56%</td>
<td>-4.81%</td>
<td>-2.97%</td>
</tr>
<tr>
<td>d %</td>
<td>-58.77%</td>
<td>-42.30%</td>
<td>-35.78%</td>
<td>46.64%</td>
</tr>
</tbody>
</table>

Table 5-8: The performance of InterProcess Communication in three different scenarios (Messages per Second)

As shown in Table 5-8, Δ% of each scenario is less than 3% of performance decrease for most sub-benchmarks of InterProcess Communication. Thus the performance impact is quite small as the number of zones increases. The apparent differences can be found in the security-enabled scenario. About -45% penalties are measured when the security mechanism for the Solaris system is enabled.
### Library/System

<table>
<thead>
<tr>
<th>Dynamic Memory Operations</th>
<th>Block Memory Operations</th>
<th>Trigonometric Functions</th>
<th>String Manipulations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global</td>
<td>trendline</td>
<td>y = -88.656x + 2214473</td>
<td>y = -17.053x + 842919</td>
</tr>
<tr>
<td>Δ %</td>
<td>-0.44%</td>
<td>-0.22%</td>
<td>-0.22%</td>
</tr>
<tr>
<td>d %</td>
<td>0.33%</td>
<td>-0.14%</td>
<td>-0.13%</td>
</tr>
<tr>
<td><strong>Non-global</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trendline</td>
<td>y = -117.34x + 2223325</td>
<td>y = -21.101x + 1431006</td>
<td>y = -21.762x + 842055</td>
</tr>
<tr>
<td>Δ %</td>
<td>-0.58%</td>
<td>-0.16%</td>
<td>-0.28%</td>
</tr>
<tr>
<td>d %</td>
<td>-0.53%</td>
<td>2.59%</td>
<td>-0.18%</td>
</tr>
<tr>
<td><strong>1 CPU core</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global</td>
<td>trendline</td>
<td>y = -106.94x + 2182522</td>
<td>y = -13.991x + 830009</td>
</tr>
<tr>
<td>Δ %</td>
<td>-1.49%</td>
<td>2.59%</td>
<td>-0.18%</td>
</tr>
<tr>
<td>d %</td>
<td>-1.50%</td>
<td>-3.12%</td>
<td>-3.69%</td>
</tr>
</tbody>
</table>

### Security

<table>
<thead>
<tr>
<th>Sort Operations</th>
<th>Signal Traps</th>
<th>Auxiliary Loops</th>
<th>Shell Script</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global</td>
<td>trendline</td>
<td>y = -0.024x + 1371.9</td>
<td>y = -11.916x + 191834</td>
</tr>
<tr>
<td>Δ %</td>
<td>-0.19%</td>
<td>-0.68%</td>
<td>0.12%</td>
</tr>
<tr>
<td>d %</td>
<td>-0.12%</td>
<td>-0.37%</td>
<td>-1.02%</td>
</tr>
<tr>
<td><strong>Non-global</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trendline</td>
<td>y = -0.0146x + 1369.9</td>
<td>y = -6.5165x + 189774</td>
<td>y = -2.4736x + 26493</td>
</tr>
<tr>
<td>Δ %</td>
<td>-0.12%</td>
<td>-0.37%</td>
<td>-1.02%</td>
</tr>
<tr>
<td>d %</td>
<td>-0.11%</td>
<td>-0.92%</td>
<td>0.47%</td>
</tr>
<tr>
<td><strong>1 CPU core</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global</td>
<td>trendline</td>
<td>y = -0.3608x + 1320.3</td>
<td>y = -355.46x + 165180</td>
</tr>
<tr>
<td>Δ %</td>
<td>-2.98%</td>
<td>-23.46%</td>
<td>-2.08%</td>
</tr>
<tr>
<td>d %</td>
<td>-5.11%</td>
<td>-23.74%</td>
<td>-2.14%</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global</td>
<td>trendline</td>
<td>y = -0.6352x + 1377.4</td>
<td>y = -88.549x + 149805</td>
</tr>
<tr>
<td>Δ %</td>
<td>-4.31%</td>
<td>-4.16%</td>
<td>4.19%</td>
</tr>
<tr>
<td>d %</td>
<td>-2.97%</td>
<td>-25.10%</td>
<td>-30.99%</td>
</tr>
</tbody>
</table>

Table 5-9: The performance of Library/System in three different scenarios
Table 5-9 shows that $\Delta\%$ of each scenario is less than 2% for most sub-benchmarks of Library/System. Thus the performance impact is quite small as the number of zones increases. However, $\Delta\%$ of Shell Script in all situations has average -15% and $\Delta\%$ of Signal Traps in the one-CPU core scenario is -23.46%.

The obvious performance difference also comes about in the one-CPU core scenario for Signal Traps with -23.74% penalties. Besides, we observe decreases of -25.10%, -30.99% and -17.29% in the security-enabled scenario using the following sub-benchmarks: Signal Traps, Auxiliary Loops and Shell Script.

➢ Algorithmic Tests

<table>
<thead>
<tr>
<th></th>
<th>Newton-Raphs on algorithm</th>
<th>3D Projection</th>
<th>Linear System</th>
<th>Integer Series</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td>Global trendline</td>
<td>y = -4.6682x + 531931</td>
<td>y = -27.025x + 1840316</td>
<td>y = -0.0306x + 991.65</td>
</tr>
<tr>
<td></td>
<td>$\Delta%$</td>
<td>-0.10%</td>
<td>-0.16%</td>
<td>-0.34%</td>
</tr>
<tr>
<td><strong>Non-global</strong></td>
<td>trendline y = -12.278x + 531638</td>
<td>y = -54.485x + 1838274</td>
<td>y = -0.0161x + 989.59</td>
<td>y = -0.027x + 82.135</td>
</tr>
<tr>
<td></td>
<td>$\Delta%$</td>
<td>-0.25%</td>
<td>-0.57%</td>
<td>-0.18%</td>
</tr>
<tr>
<td></td>
<td>d %</td>
<td>-0.13%</td>
<td>-0.19%</td>
<td>-0.13%</td>
</tr>
<tr>
<td><strong>1 CPU core</strong></td>
<td>Global trendline</td>
<td>y = -98.412x + 525500</td>
<td>y = 341.23x + 1756282</td>
<td>y = -0.227x + 949.5</td>
</tr>
<tr>
<td></td>
<td>$\Delta%$</td>
<td>-2.04%</td>
<td>2.12%</td>
<td>-2.61%</td>
</tr>
<tr>
<td></td>
<td>d %</td>
<td>-2.18%</td>
<td>-3.47%</td>
<td>-5.34%</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Global trendline</td>
<td>y = -170.29x + 537215</td>
<td>y = -313.49x + 1839111</td>
<td>y = -0.3778x + 996.41</td>
</tr>
<tr>
<td></td>
<td>$\Delta%$</td>
<td>-3.49%</td>
<td>-1.89%</td>
<td>-3.49%</td>
</tr>
<tr>
<td></td>
<td>d %</td>
<td>-1.24%</td>
<td>-1.15%</td>
<td>-2.17%</td>
</tr>
</tbody>
</table>

Table 5-10: the performance of Algorithmic Tests in three different scenarios

Table5-10 shows that $\Delta\%$ and d % are less than 3% of performance decrease for most sub-benchmarks of Algorithmic Tests in all the three scenarios. Thus,
the performance impact as increasing numbers of zones and difference in various scenarios both are quite small for Algorithmic Tests.

In conclusion, the performance impact as increasing numbers of zones is quite small except for the sub-benchmarks of Memory and Process Management with an average of 10% of performance decrease. The performance of the non-global zone in the baseline scenario is similar to the baseline global zone. Also, most performance of the global zone in the one-CPU core scenario is the same as for the baseline global zone with the following exceptions. The performance of Memory and Process Management shows about 15% of performance decrease, as well as for sub-benchmark Arithmetical Additions, File Creations and Closes, and Signal Traps. Finally, the performance of the global zone in the security-enabled scenario obviously declines in some fields compared to the baseline global zone. There is average of -30%, -15% and -40% for Disk/Filesystem I/O, Memory and Process Management, and Interprocess Communication respectively.

5.2.2 Performance of running benchmarks concurrently in different zones

In this part, we present and discuss the results of running multiple instances of the benchmark at the same time in different zones. Our evaluation is divided into two branches: one is used to gather the baseline performance of Solaris Zones; the other is implemented in the security configuration-enabled scenario.

The maximum number of zones in the system for the baseline concurrent evaluation is 44 zones (non-global zones). Most sub-benchmarks cannot work correctly from 45 zones running in the system, because necessary resources are used up. For the security configuration-enabled scenario, we measure the concurrent performance increasing to 38 zones (labeled zones).
To compare the results with running a single instance of the benchmark in the system, we calculate the total value of performance when there are \( n \) non-global zones in the system: \( S_n := \sum_{i=1}^{n} R_{in} \). \( R_{in} \) denotes the result of the benchmark running in the \( i^{th} \) non-global zone when \( n \) non-global zones exist. Then, the performance score relative to a single instance of the benchmark can be described as \( D_n = S_n / R_{11} \).

In the following part, we present and comment the results of the two scenarios from seven aspects. The parameters and sub-benchmarks are the same as the sequence in section 5.2.1.

- Arithmetic

In Figure 5-4, we show the aggregate arithmetic performance that the Solaris system achieves when running 1 to 44 instances of zones on the system. When a second zone is added, full utilization of the computing resources almost doubles the total performance. Increasing the number of zones further causes some reduction in aggregate performance. Aggregate scores running multiple benchmark instances decrease slowly from 2 to 1.5 as the number of zones increases. Therefore, the aggregate performance in this case is better than the one of a single instance.

Figure 5-5 demonstrates the relevant performance in the security configuration-enabled scenario. Figure 5-6 shows the gradient of relevant regression lines for aggregate scores in both scenarios. The difference of each sub-benchmark is no more than 5% between the two scenarios. Thus, we can easily find that the trend of aggregate performance in the security-enabled scenario is similar to the baseline scenario.
Figure 5-4: The overall performance of Arithmetic for up to 44 concurrent zones in the baseline Scenario

Figure 5-5: The overall performance of Arithmetic for up to 38 concurrent zones in the security-enabled scenario
Figure 5-6: Comparison of overall performance between the baseline scenario and the security-enabled scenario (Arithmetic performance)

Memory and Process Management

Figure 5-7 shows the aggregate performance of Memory and Process Management in the baseline scenario. Aggregate scores of Memory Allocations and Memory Allocations & Pages reduce gently from about 1.8 to 1.3 with increasing numbers of zones. However, the performance of Program Loads and Task Creations is quite different. When a second zone is added, an extra 50% gain is achieved. Increasing the number of zones causes further rapid decrease. As a result, the aggregate performance of six or more zones is always worse than the performance of a single instance.

Figure 5-8 demonstrates the aggregate performance of Memory and Process Management in the security-enabled scenario. From Figure 5-9, we can see that the gradient of each sub-benchmark is almost the same between the two scenarios. Thus, we can find that the trend of aggregate
performance in the security-enabled scenario is similar to the baseline scenario.

Figure 5-7: The overall performance of Memory and Process Management for up to 44 concurrent zones in the baseline scenario
Figure 5-8: The overall performance of Memory and Process Management for up to 38 concurrent zones in the security-enabled scenario

Figure 5-9: Comparison of overall performance between the baseline scenario and the security-enabled scenario (Memory and Process Management performance)
Disk/Filesystem I/O

From Figure 5-10, we can see the aggregate performances of Disk/Filesystem I/O in the baseline scenario. The performance fluctuates around score 1.5 with a slight decline as the number of zones increases. It demonstrates that an extra 50% gain can be achieved when running multiple Disk/Filesystem I/O applications concurrently in different zones than in the single instance. Particularly, the aggregate performance of Directory Operations benefits more than average. Aggregate scores achieve 2.5 when running 5 to 22 concurrent zones and after 23 zones the scores stay on 2.

Besides, Figure 5-11 demonstrates that the aggregate performance of Disk/Filesystem I/O in the security-enabled scenario. From Figure 5-12, we can see that the gradient difference of each sub-benchmark is small between two scenarios. Thus, we can find that the trend of aggregate performance in the security-enabled scenario is similar to the baseline scenario.
Figure 5-10: The overall performance of Disk/FileSystem I/O for up to 44 concurrent zones in the baseline scenario

Figure 5-11: The overall performance of Disk/FileSystem I/O for up to 38 concurrent zones in the security-enabled scenario
Figure 5-12: Comparison of overall performance between the baseline scenario and the security-enabled scenario (Disk/Filesystem I/O performance)

- **Function Calls**

  In Figure 5-13, we show the aggregate performance of Function Calls in the baseline scenario. When a second zone is added, full utilization of the computing resources almost doubles the total performance. Further increasing the number of zones causes some reduction in aggregate performance. Aggregate scores running multiple benchmark instances decrease slowly from 2 to 1.5 as the number of zones grows up. Therefore, the aggregate performance in this case is better than the one of a single instance.

  Figure 5-14 demonstrates the relevant performance in the security configuration enabled scenario. From Figure 5-15, we can see that the gradient of each sub-benchmark is no more than 4% between two scenarios.
Thus, we can find that the trend of aggregate performance in the security-enabled scenario is similar to the baseline scenario.

Figure 5-13: The overall performance of Function Calls for up to 44 concurrent zones in the baseline scenario
Figure 5-14: The overall performance of Function Calls for up to 38 concurrent zones in the security-enabled scenario

Figure 5-15: Comparison of overall performance between the baseline scenario and the security-enabled scenario (Function Calls performance)
InterProcess Communication

From Figure 5-16, we can see the aggregate performance of InterProcess Communication in the baseline scenario. The throughput of TCP/IP and UDP/IP fluctuates around score 1.5 from running two instances of zone. It demonstrates that an extra 50% gain can be achieved when running multiple TCP/IP or UDP/IP applications concurrently in different zones than the single instance. The trend of FIFO is same as the situation of TCP/IP and UDP/IP. When exceeding 25 zones in the system, the throughput falls to approximate the value of a single instance. The aggregate throughput of Pipe benefits more than average. It doubles the throughput when a second zone is added. And aggregate scores rise gradually to 3 as the number of zones increases further.

Figure 5-17 presents the aggregate performance of InterProcess Communication in the security configuration enabled scenario. From Figure 5-18, we can see that the gradient difference of each sub-benchmark is small between two scenarios. Thus, we can find that the trend of aggregate performance in the security-enabled scenario is similar to the baseline scenario.
Figure 5-16: The overall performance of InterProcess Communication for up to 44 concurrent zones in the baseline scenario

Figure 5-17: The overall performance of InterProcess Communication for up to 38 concurrent zones in the security-enabled scenario
Figure 5-18: Comparison of overall performance between the baseline scenario and the security-enabled scenario (InterProcess Communication performance)

- **Library/System**
  In Figure 5-19, we show the aggregate performance of Library/System in the baseline scenario. When a second zone is added, full utilization of the computing resources almost doubles the total performance. Most aggregate scores running multiple benchmark instances decrease slowly from 2 to 1.5 as the number of zones grows up. However, the aggregate performance of Auxiliary Loops benefits more than average. Its aggregate scores stay above 2 for most situations. The aggregate performance of Shell Script decreases as the number of zones increases. Its aggregate performance of six or more zones is always worse than the performance of a single instance.

Figure 5-20 demonstrates the relevant performance in the security configuration-enabled scenario. From Figure 5-21, we can see that the
gradient difference of each sub-benchmark is similar to the baseline scenario without one sub-benchmark. It shows that the aggregate performance of Auxiliary Loops no longer has gains that are greater than average. Its aggregate scores decline from 2 to 1.5 in the security-enabled scenario.

Figure 5-19: The overall performance of Library/System for up to 44 concurrent zones in the baseline scenario
Figure 5-20: The overall performance of Library/System for up to 38 concurrent zones in the security-enabled scenario

Figure 5-21: Comparison of overall performance between the baseline scenario and the security-enabled scenario (Library/System performance)
- **Algorithmic Tests**

In Figure 5-22, we show the aggregate performance of Algorithmic Tests in the baseline scenario. When a second zone is added, full utilization of the computing resources almost doubles the total performance. Most aggregate scores running multiple benchmark instances decrease slowly from 2 to 1.5 as the number of zones grows up. However, the throughput of the *Taylor Series* benchmark fluctuates between 2 and 3.

Additionally, Figure 5-23 demonstrates the aggregate performance of Algorithmic Tests in the security-enabled scenario. From Figure 5-24, we can see that the gradient difference of each sub-benchmark is small between two scenarios. Thus, we can find that the trend of the aggregate performance in the security-enabled scenario is similar to the baseline scenario.

![Figure 5-22: The overall performance of Algorithmic Tests for up to 44 concurrent zones in the baseline scenario](image-url)
Figure 5-23: The overall performance of Algorithmic Tests for up to 38 concurrent zones in the security-enabled scenario

Figure 5-24: Comparison of overall performance between the baseline scenario and the security-enabled scenario (Algorithmic Tests performance)
In conclusion, the aggregate performance when running multiple instances of the benchmark in different zones is better than that of a single instance. The maximum of the aggregate performance is achieved when a second zone is added, which can be attributed to full utilization of hardware resources. Then, increasing the number of zones further causes some reduction in aggregate performance, because of the context switching and disk head movement. However, the aggregate performance of some sub-benchmarks is not better than the performance of a single instance as the number of zones increases. We conjecture that the reason of this phenomenon is that the system does not have enough memory to finish its tasks efficiently when benchmarks are run concurrently in different zones.

5.3 Discussion

In this section, we present some discussions based on the results mentioned above. Generally speaking, we discuss the evaluation results with respect to two aspects: system scalability and system performance.

5.3.1 Discussion based on system scalability

For system scalability, the results indicate that the creation time of Solaris Zones is influenced when the number of zones in one system increases. According to the results, it costs approximately 5 minutes to create an additional non-global zone if the total number of zones is less than 86. From the 86th non-global zone on, the creation time of each zone remains at approximately 13 minutes. However, from the 110th non-global zone on, the creation time increases sharply to an unacceptable value. This phenomenon directly reveals one ‘bottle neck’ of our test bed, which is system memory. Because each running zone occupies part of the memory, the system does not have enough memory to create a zone efficiently after 109 non-global zones have been created. Therefore, considering a value of up to 15 minutes...
acceptable, we use 109 non-global zones as the maximum number of zones in our performance evaluation part.

5.3.2 Discussion based on system performance

For system performance, we discuss the results based on the two major evaluation circumstances, which are running the benchmark in a single zone and running the benchmarks concurrently in different zones. In the case of running the benchmark in a single zone, two important parameters are involved to measure performance influences. They are system performance impact as the number of zones increases (Δ%) and system performance difference compared with the baseline metrics (d%). In the case of running benchmarks concurrently in different zones, we aggregate the benchmark results relative to single instance performance.

5.3.2.1 Running the benchmark in a single zone

When running the benchmark in a single zone, most values of d% and Δ% are very small, which means most performance perturbations are slight. However, there are several exceptions, in which the values of d% and Δ% are larger and the system performance is affected. According to the testing results of Disk/Filesystem I/O, the values of d% in the security enable scenario are about 30% averagely. Similarly, all values of d% in InterProcess Communication test and some values of d% in Library/System test are considerable in the security enable scenario. In the above three cases, the system performance is seriously influenced when security mechanisms are enabled in the system. The security mechanism used in our test is implemented by Solaris Trusted Extensions, which applies its MAC mechanisms. Thus, the system spends more time on calculating access control decisions, which is the reason of the observed decrease in performance. Moreover, most values of d% and Δ% in the Memory and Process Management are not small. As
mentioned above, the system memory is getting consume when new zones are in the running state or running new processes. In this case, the system performance decreases due to lack of memory.

5.3.2.2 Running benchmarks concurrently in different zones

When running benchmarks concurrently in different zones, we use 44 zones as the maximum number of zones for the baseline scenario and 38 for the security enable scenario in our evaluation. The reason is that system resources, such as memory and disk, cannot afford more zones to work simultaneous. According to the results, in each type of the evaluation, the overall trends of most sub-benchmarks are similar within one scenario. Moreover, the aggregate performance of one sub-benchmark in the baseline scenario and the security enable scenario also follows a similar trend. On our hardware configuration, in most cases, the maximum value of aggregate performance is achieved when two zones are running simultaneously in the Solaris system. This phenomenon is attributed to full utilization of hardware resources. Then, along with increasing the number of zones, the aggregate performance is gently reduced with some fluctuations. We conjecture that this phenomenon is attributable to the increase of context switching. In our test, the aggregate scores do not fall below 1 in most cases, which means that the performance of running benchmarks in multiple zones is better than running the benchmark in a single zone.

However, compared to the general trends we discussed in our test, in the Memory and Process Management test, the aggregate scores of Program Loads and Task Creations fall below 1 after 6 or more zones running benchmarks simultaneously in the system. Similarly, in the Library/System test, the aggregate performance of the Shell Script sub-benchmark is less then 1 for 6 or more zones. The reason of these phenomena is that the system does
not have enough memory to finish its tasks efficiently when benchmarks are run concurrently in 6 or more zones.

5.4 Summary

In this chapter, we first present the results of our Solaris Zones performance evaluation. The evaluation is carried out by running the AIM benchmark in three scenarios. In the first scenario we measure - as a baseline - the performance of Solaris Zones on a 2-CPU core machine in the standard configuration that is distributed as part of the Solaris OS. In the second scenario we investigate the influence of the number of CPU cores, by using the same tests on just a single CPU core. In the third scenario we evaluate the performance in the presence of a security configuration known as Solaris Trusted Extensions.

The results are divided into two types, system scalability and system performance. Then, we presented some discussions based on these results. Except some individual differences, the performance impacts for increasing numbers of zones are slight compared to running the benchmark in a single zone. However, in some cases, system performance may be worse than baseline performance in the security-enabled scenario. When running benchmarks concurrently in multiple zones, most aggregate performance is better than the performance of a single instance. However, due to hardware resource limitations, some aggregate performance evaluations do not exceed the performance of a single instance.

Nevertheless, we do not compare our evaluation result with the outcomes in the existing research. The reason is that the test bed, test tool and test protocol are different in our evaluation compared to the existing research.
Chapter 6
Conclusion and Further Research

In this chapter, we first present the conclusion of this thesis. Then, we indicate the methodology limitations in our test. Finally, some ideas of the future work are described.

6.1 Conclusion

The purpose of this thesis work is to measure and evaluate the performance impacts of Solaris Zones on the Solaris OS under different system configurations. In Chapter 2, two research problems are proposed. The first research problem is to examine how well the Solaris OS scales with respect to an increasing number of zones. Besides, we also intend to uncover the maximum number of zones that the Solaris OS can support simultaneously with acceptable performance. The second research problem is to measure the performance influence under different configurations.

In Chapter 3, we investigate the evaluation methods of existing OS-level virtualization technologies. In Chapter 4, based on the two research problems, we present the test bed, test tool, test protocol and test description respectively. In our test, the test bed consists of two computers with the same hardware
configuration. Both of them are running the Solaris OS (SunOS 5.11 snv_44 i86pc). Then, we use the AIM benchmark as our test tool. The main advantage of the AIM benchmark is its applicability for measuring the performance of configuration changes on a single system. The protocol used in our test consists of two parts, which are scalability and performance. For the scalability part, we examine the Solaris system’s zone installation ability by recording the creation time of each zone. This test aims to solve the first research problem. For the performance part, we inspect the system performance influence in three scenarios. In the first scenario we measure - as a baseline - the performance of Solaris Zones on a 2-CPU core machine in the standard configuration that is distributed as part of the Solaris OS. In the second scenario we investigate the influence of the number of CPU cores. In the third scenario we evaluate the performance in the presence of a security configuration known as Solaris Trusted Extensions. These tests aim to solve the second research problem.

Finally, we present and analyze the evaluation results in Chapter 5. Firstly, we discover that the creation time of additional zones sharply increases to an unacceptable value (more than one hour per zone) from the 110th non-global zone on. Therefore, running up to 109 non-global zones simultaneously is the utmost situation under our system configuration, which successfully solves the research problem of scalability. Secondly, we discover that most performance influences are slight and less than 2% of performance decrease in each scenario when running the benchmark in a single zone and continuously creating new zones. However, after enabling the security mechanism, the performance declines in Disk/Filesystem I/O test, InterProcess Communication test. When running the benchmark concurrently in multiple zones, most aggregate performance is better than the performance of a single instance. However, some aggregate performance cannot exceed the performance of a single instance, such as Program Loads, Task Creations and so on.
According to the testing results, our evaluation successfully solves the two proposed research problems. Therefore, our test towards the Solaris OS can be considered as a successful test. Our evaluation contributes the performance influence of Solaris Zones on the Solaris OS in specific scenarios. In other words, our tests quantify the performance impact of varying the number of used zones, the number of CPU cores, and basic security configurations of Solaris Trusted Extensions.

### 6.2 Methodology Limitations

One methodology limitation in our test is the limited availability of a greater range of hardware configurations, such as varying the number of CPUs and the size of main memory and disk. When examining the scalability of the Solaris OS, the creation time for additional zones sharply increases to an unacceptable value from the 110th non-global zone on. The reason of this phenomenon is not a flaw in the design or implementation of the Solaris OS but the limited amount of memory of our evaluation platform. We expect that if the memory size changes, the maximum number of zones with acceptable performance will also change. Therefore, it is hard to generalize our test result to other system configuration.

In our performance evaluation, we use the AIM benchmark as our test tool. The AIM benchmark consists of 60 sub-benchmarks, which can be classified into seven categories as the following: Arithmetic, Memory and Process Management, Disk/Filesystem I/O, Function Calls, Interprocess Communication, Library/System, and Algorithmic Tests. The tests in some categories cause an OS performance overhead that is noticeable, while others have little influence on the OS performance. Additionally, each sub-benchmark of the AIM benchmark is running in a single thread. We expect that we would get an even more comprehensive picture of the performance of Solaris Zones if
the AIM benchmark contained a category with sub-benchmarks that evaluate multi-threading system behavior.

Another methodology limitation exists in our test protocol. During our test, we carry out each evaluation only once because of the arduous data processing and limited thesis work duration. However, it is not rigorous enough to perform each test only once. To minimize the statistical bias, each evaluation should be carried out repeatedly, followed by an averaging of the results.

Finally, we used only a single configuration in the security-enabled scenario. In our evaluation, the system operates a single-level session for comparing different labels. Therefore, we can only quantify the performance penalty of the security mechanism under this configuration of single-level session. We are not able to state how the performance changes when other configurations of Solaris Trusted Extensions are used.

6.3 Future Work

According to these methodology limitations of our test, the following investigations could be carried out in the future to improve on the results of our work.

In our work, the influence of limited hardware resources is considered to be a big obstacle to generalize our test results. Even if we change the system configuration and repeat the tests, it still cannot solve this problem. Therefore, one future work could aim at determining the relationship between system configurations and performance influence by using some artificial intelligence tools, such as neural networks.
Moreover, because some sub-benchmarks are not significant toward our result, we can only preserve the significant ones, which are Disk/FileSystem I/O, Memory and Process Management, Interprocess Communication, and Library/System sub-benchmarks. Besides, to eliminate the delay caused by running multiple instances of these single-thread benchmarks simultaneously, it would be useful to have sub-benchmarks that evaluate multi-threading system behavior.

Another avenue future work could take is to repeat our tests to minimize the statistical bias in the test results. Moreover, additional results from repeated tests also can be used to train neural networks.

To evaluate the performance of the security mechanism when running in the Solaris OS, we can extend the security-enabled scenario by examining the performance penalty of Solaris Zones under different security configurations. For instance, we could compare the performance of a single-level session and a multilevel session. In other words, the future work could focus on a fine-grained analysis of the performance differences of a variety of security configurations.

Finally, we examine the scalability and performance of Solaris Zones under different configurations. To understand the capability of Solaris Zones, not only should we comprehensively evaluate this OS virtualization technology, but also identify the performance characteristics of other OS virtualization technologies under the same testing protocol and on identical hardware. Therefore, it is recommended to compare the evaluation results among different OS virtualization technologies.
Reference


Appendix

Scripts of evaluations

a) The evaluation script for the global zone in the baseline scenario

```bash
#!/bin/bash
for ((i=1; i<=130; i++))
do
echo yuan$i
mkdir /usr/local/yuan$i
mkdir /opt/local/yuan$i
mkdir /opt/sfw/yuan$i
zonecfg -z yuan$i "create ;
    set zonepath=/export/home/yuan$i ;
    set autoboot=true ;
    add fs ; set dir=/usr/local/yuan$i ;
    set special=/opt/local/yuan$i ; set type=lofs ;
    end ;
    add inherit-pkg-dir ; set dir=/opt/sfw/yuan$i ;
    end ;
    add net ; set address=192.168.25.$i ;
    set physical=bge0 ; end "
zoneadm -z yuan$i install
zoneadm -z yuan$i boot
cd /usr/local/sw/aim9
./singleuser < input
rpt9 suite9.ss /usr/local/sw/results/result$i.ps
mv suite9.ss /usr/local/sw/results/suite9/suite$i.ss
mv logfile.suite9 /usr/local/sw/results/logfile/logfile$i.suite9
done
```
b) The evaluation script for non-global zone in the baseline scenario

```bash
#!/bin/bash
for ((i=1;i<=110;i++))
do
  if [ $i -eq 1 ]; then
    echo yuan1
    zlogin yuan1 /usr/local/sw/ab
    mv /export/home/yuan1/root/sw/aim9/suite9.ss
    /export/home/yuan1/root/sw/results/result$i.ss
  else
    echo yuan$i
    mkdir /usr/local/yuan$i
    mkdir /opt/local/yuan$i
    mkdir /opt/sfw/yuan$i
    zonecfg -z yuan$i "create ;
    set zonepath=/export/home/yuan$i ;
    set autoboot=true ;
    add fs ; set dir=/usr/local/yuan$i ;
    set special=/opt/local/yuan$i ; set type=lofs ;
    end ;
    add inherit-pkg-dir ; set dir=/opt/sfw/yuan$i ;
    end ;
    add net ; set address=192.168.25.$i ;
    set physical=bge0 ; end "
    zoneadm -z yuan$i install
    zoneadm -z yuan$i boot
    zlogin yuan1 /usr/local/sw/ab
    mv /export/home/yuan1/root/sw/aim9/suite9.ss
    /export/home/yuan1/root/sw/results/result$i.ss
    rm /tmp/*.*
  fi
done
```
c) The evaluation script for aggregate performance in the baseline scenario

```bash
#!/bin/bash
for ((i=1; i<=50; i++))
do
  echo yuan$i
  mkdir /usr/local/yuan$i
  mkdir /opt/local/yuan$i
  mkdir /opt/sfw/yuan$i
  zonecfg -z yuan$i "create ;
  set zonepath=/export/home/yuan$i ;
  set autoboot=true ;
  add fs ; set dir=/usr/local/yuan$i ;
  set special=/opt/local/yuan$i ; set type=lofs ;
  end ;
  add inherit-pkg-dir ; set dir=/opt/sfw/yuan$i ;
  end ;
  add net ; set address=192.168.22.$i ;
  set physical=bge0 ; end "
  zoneadm -z yuan$i install
  cp /usr/local/sw/sysidcfg /export/home/yuan$i/root/etc/
  zoneadm -z yuan$i boot
  sleep 1200
  mkdir /export/home/yuan$i/root/sw
  cp /usr/local/sw/aim.tar /export/home/yuan$i/root/sw/aim.tar
  cd /export/home/yuan$i/root/sw
  tar xvf aim.tar
  for ((j=$i; j>0; j--))
    do
      zlogin yuan$j /usr/local/sw/ab &
    done
  sleep 700
  for ((j=$i; j>0; j--))
    do
      cd /export/home/yuan$i/root/sw/aim9
      rpt9 suite9.ss /usr/local/sw/results/result$i$j.ps
      mv suite9.ss /usr/local/sw/results/suite9/suite$i$j.ss
      mv logfile.suite9
      /usr/local/sw/results/logfile/logfile$i$j.suite9
    done
  done
```
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Yuan Xu
March 12, 2007