[Summary 1](#_TOC_250011)

[Context and Scope 2](#_TOC_250010)

[Context 2](#_TOC_250009)

[Goals 2](#_TOC_250008)

[Non-Goals: 2](#_TOC_250007)

[Design 2](#_TOC_250006)

[Overview 2](#_TOC_250005)

[Details 3](#_TOC_250004)

System Context Diagram 3

Key constraints 3

[APIs 3](#_TOC_250003)

Data storage 3

Prototypes, if any 3

[Cross Cutting Concerns 3](#_TOC_250002)

Security 3

Privacy 3

Observability 3

Governance 3

[Alternatives Considered 3](#_TOC_250001)

[References 3](#_TOC_250000)

Software Design Document

# Summary

A Single System Image (SSI) is an abstraction that provides the illusion that a multicomputer or cluster is a single machine. There are individual instances of the Operating Systems (OSs) running on each node of a multicomputer, processes working together are spread across multiple nodes, and files may reside on multiple disks. An SSI provides a unified view of this collection to users, programmers, and system administrators. This unification makes a system easier to use and more efficient to manage.

# Context and Scope

## Context

* System requirements link
* Specific facts about the situation the new software is for.
* Objective analysis situating the proposed software in the existing system.

## Goals

1. *Single entry point:* A user can connect to the cluster as a virtual host (e.g., telnet beowulf.myinstitute.edu), although the cluster may have multiple physical host nodes to serve the login session. The system transparently distributes the user’s connection requests to different physical hosts to balance the load.
2. *Single user interface:* The user should be able to use the cluster through a single GUI. The interface must have the same look and feel as the one available for workstations (e.g., Solaris Open Win or Windows NT GUI).
3. *Single process space:* All user processes, no matter on which nodes they reside, have a unique cluster-wide process ID. A process on any node can create child processes on the same or different node (through a UNIX fork). A process should also be able to communicate with any other process (through signals and pipes) on a remote node. Clusters should support globalized process management and allow the management and control of processes as if they are running on local machines.
4. *Single memory space:* Users have an illusion of a big, centralized main memory, which in reality may be a set of distributed local memories. Software DSM approach has already been used to achieve single memory space on clusters. Another approach is to let the compiler distribute the data structure of an application across multiple nodes. It is still a challenging task to develop a single memory scheme that is efficient, platform independent, and able to support sequential binary codes.
5. *Single I/O space (SIOS):* This allows any node to perform I/O operations on local or remotely located peripheral or disk devices. In this SIOS design, disks associated to cluster nodes, network-attached RAIDs, and peripheral devices form a single address space.
6. *Single-file hierarchy:* On entering into the system, the user sees a single, huge file system image as a single hierarchy of files and directories under the same root directory that transparently integrates local and global disks and other file devices. Examples of single-file hierarchy include NFS, AFS, xFS, and Solaris MC Proxy.
7. *Single virtual networking:* This means that any node can access any network connection throughout the cluster domain even if the network is not physically connected to all nodes in the cluster. Multiple networks support a single virtual network operation.
8. *Single job management system:* Under a global job scheduler, a user job can be submitted from any node to request any number of host nodes to execute it. Jobs can be scheduled to run in either batch, interactive, or parallel modes. Examples of job management systems for clusters include GLUnix, LSF, and CODINE.
9. *Single control point and management:* The entire cluster and each individual node can be configured, monitored, tested, and controlled from a single window using single GUI tools, much like an NT workstation managed by the task manager tool.
10. *Checkpointing and process migration:* Checkpointing is a software mechanism to periodically save the process state and intermediate computing results in memory or disks. This allows the rollback recovery after a failure. Process migration is needed in dynamic load balancing among the cluster nodes and in supporting checkpointing.

## Non-Goals:

* What won’t this software do ?
* (E.g. will not provide atomic writes)

# Design

## Overview

SSI can be implemented in one or more of the following levels:

* hardware,
* operating system (so-called “underware”) (Walker and Steel, 1999a),
* middleware (runtime subsystems),
* application.

A good SSI is usually obtained by cooperation between all these levels as a lower level can simplify the implementation of a higher one.

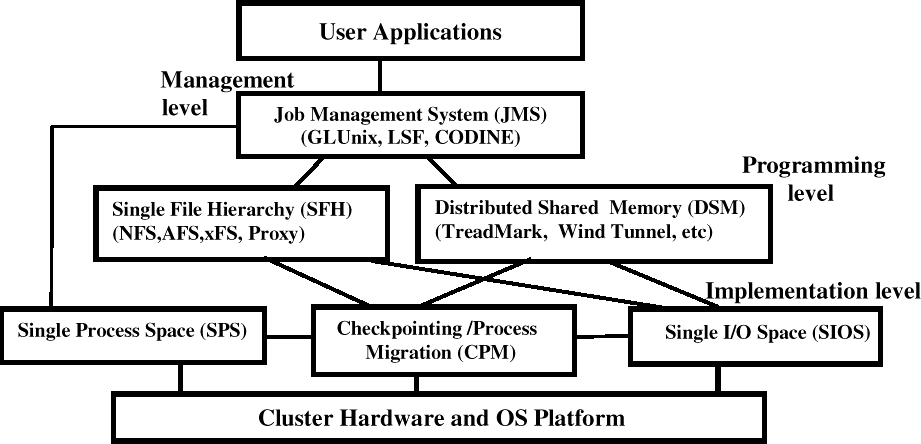
## Details

* 1. **HARDWARE LEVEL**

## **OPERATING SYSTEM LEVEL**

* 1. **MIDDLEWARE LEVEL**
  2. **APPLICATION LEVEL**

System Context Diagram



Key constraints

The following constraints can be identified

1. The hardware-level SSI can offer the highest level of transparency, but due to its rigid architecture, it does not offer the flexibility required during the extension and enhancement of the system.
2. The kernel-level approach offers full SSI to all users (application developers and end users). However, kernel-level cluster technology is expensive to develop and maintain, as its market share is or will be probably limited, and it is difficult to keep pace with technological innovations emerging into mass-market operating systems.

An application-level approach helps realize SSI partially and requires that each application be developed as SSI aware separately. A key advantage of application-level SSI compared with the kernel level is that it can be realized in stages, and the user can benefit from it immediately, but in the kernel-level approach, unless all components are specifically developed to support SSI, it cannot be used or released to the market. Due to this, the kernel-level approach appears as a risky and economically nonviable approach.

1. The middleware approach is a compromise between the other two mechanisms used to provide SSI. In some cases, such as in PVM, each application needs to be implemented using special APIs on a case-by-case basis. This means that there is a higher cost of implementation and maintenance; otherwise, the user cannot get any benefit from the cluster.

# Cross Cutting Concerns

SSI can greatly enhance the acceptability and usability of clusters by hiding the physical existence of multiple independent computers by presenting them as a single, unified resource. SSI can be realized either using hardware or software techniques; each has its own advantages and disadvantages. The middleware approach appears to offer an economy of scale compared with other approaches, although it cannot offer full SSI like the OS approach. In any case, the designers of software (system or application) for clusters must always consider SSI (transparency) as one of their important design goals in addition to scalable performance and enhanced availability

# Alternatives Considered

Comparing to [Parallel Virtual Machine](https://en.wikipedia.org/wiki/Parallel_Virtual_Machine)  and [Message Passing Interface](https://en.wikipedia.org/wiki/Message_Passing_Interface) SSI is much better when it comes to

# References

* This design document is roughed out from these ideas

https://web.archive.org/web/20210909084007/https://[www.industrialempathy.com/posts/](http://www.industrialempathy.com/posts/)

design-docs-at-google/

-