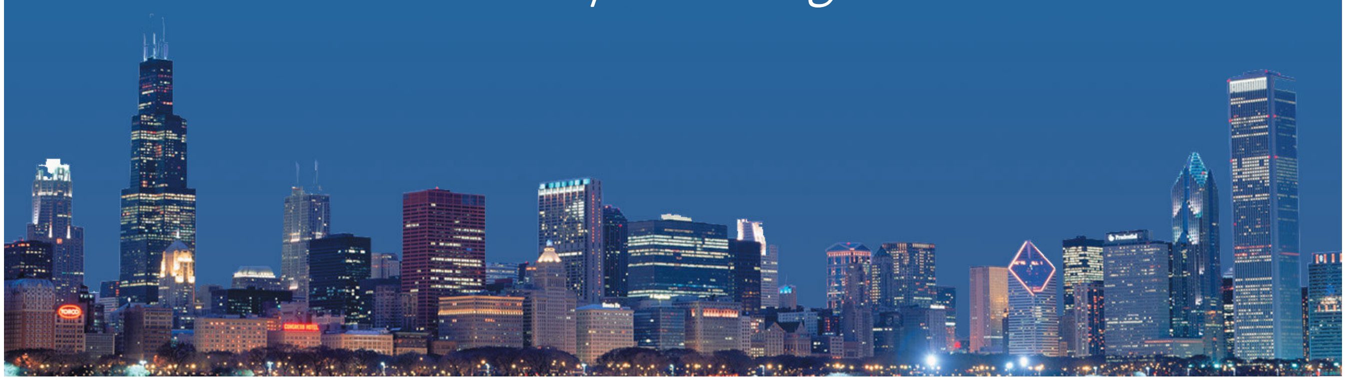
HP WORLD 2004
Solutions and Technology Conference & Expo

# OpenSSI (Single System Image) Linux Cluster Project

openssi.org



**Jeff Edlund**
**Senior Principle Solution Architect NSP**

**Bruce Walker**
**Staff Fellow – Office of Strategy & Technology**

# Agenda

- What are today's clustering strategies for Linux
- Why isn't failover clustering enough
- What is Single System Image (SSI)
- Why is SSI so important
- OpenSSI Cluster Project Architecture
- Project Status

# Many types of Clusters

- High Performance Clusters
  - Beowulf; 1000 nodes; parallel programs; MPI
- Load-leveling Clusters
  - Move processes around to borrow cycles (eg. Mosix)
- Web-Service Clusters
  - LVS/Piranah; load-level tcp connections; replicate data
- Storage Clusters
  - GFS; parallel filesystems; same view of data from each node
- Database Clusters
  - Oracle Parallel Server;
- High Availability Clusters
  - ServiceGuard, Lifekeeper, Failsafe, heartbeat, failover clusters
- Single System Image Clusters

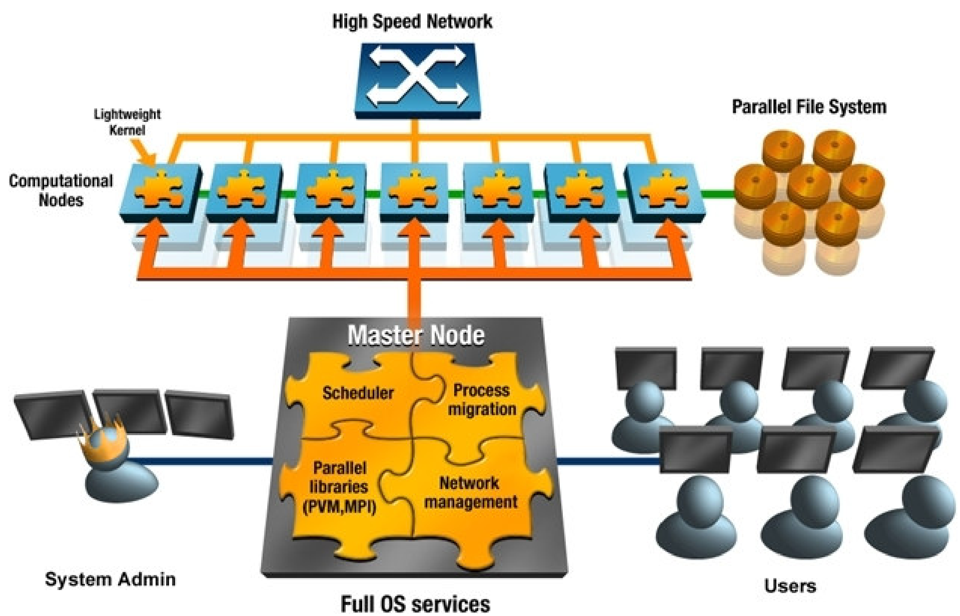
# Who is Doing SSI Clustering?

- Outside Linux:
  - Compaq/HP with VMSClusters, TruClusters, NSK, and NSC
  - Sun had “Full Moon”/Solaris MC (now SunClusters)
  - IBM Sysplex ?
- Linux SSI:
  - Scyld - form of SSI via Bproc
  - Mosix - form of SSI due their homenode/process migration technique and looking at a single root filesystem
  - Polyserve - form of SSI via CFS (Cluster File System)
  - QClusters – SSI through software / middleware layer
  - RedHat GFS – Global File System (based on Sistina)
  - Hive Computing – Declarative programming model for “workers”
  - OpenSSI Cluster Project – SSI project to bring all attributes together

# Scyld - Beowulf

## Bproc (used by Scyld):

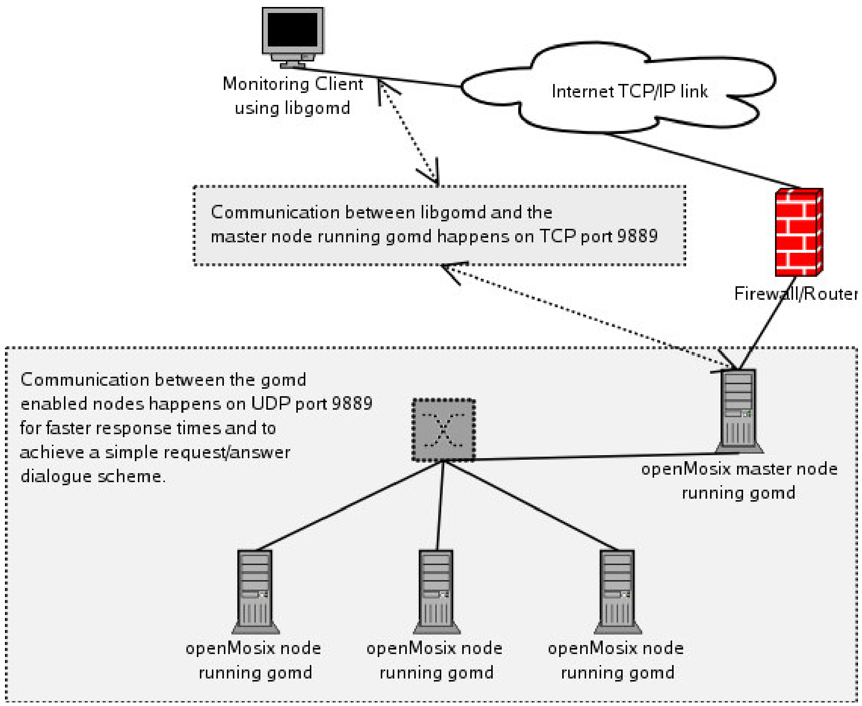
- process-related solution
- master node with slaves
- initiate process on master node and explicitly "move", "rexec" or "rfork" to slave node
- all files closed when the process is moved
- master node can "see" all the processes which were started there
- moved processes see the process space of the master (some pid mapping)
- process system calls shipped back to the master node (including fork)
- other system calls executed locally but not SSI



# Mosix

## Mosix / OpenMosix:

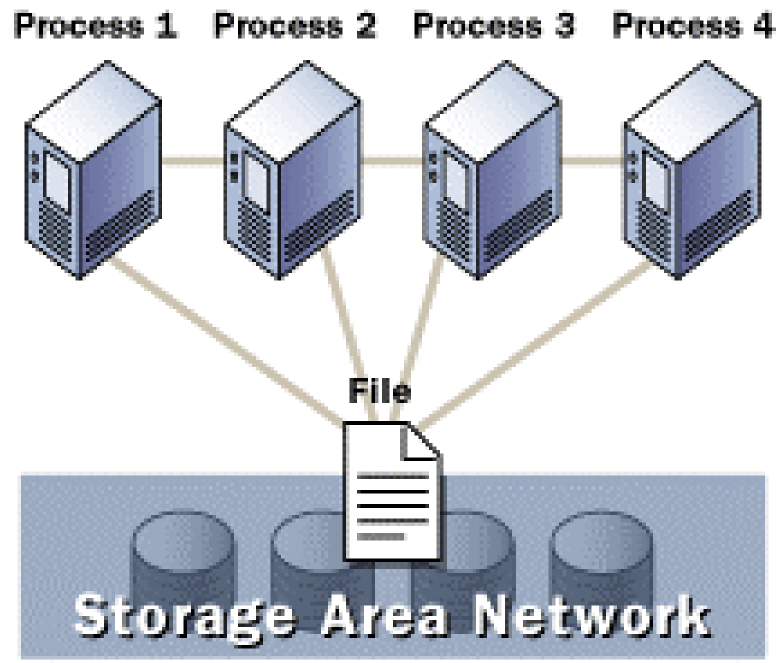
– home nodes with slaves
– initiate process on home node and transparently migrate to other nodes
– home node can “see” all and only all processes started there
– moved processes see the view of the home node
– most system calls actually executed back on the home node
– DFSA helps to allow I/O to be local to the process



# PolyServe

Matrix Server:

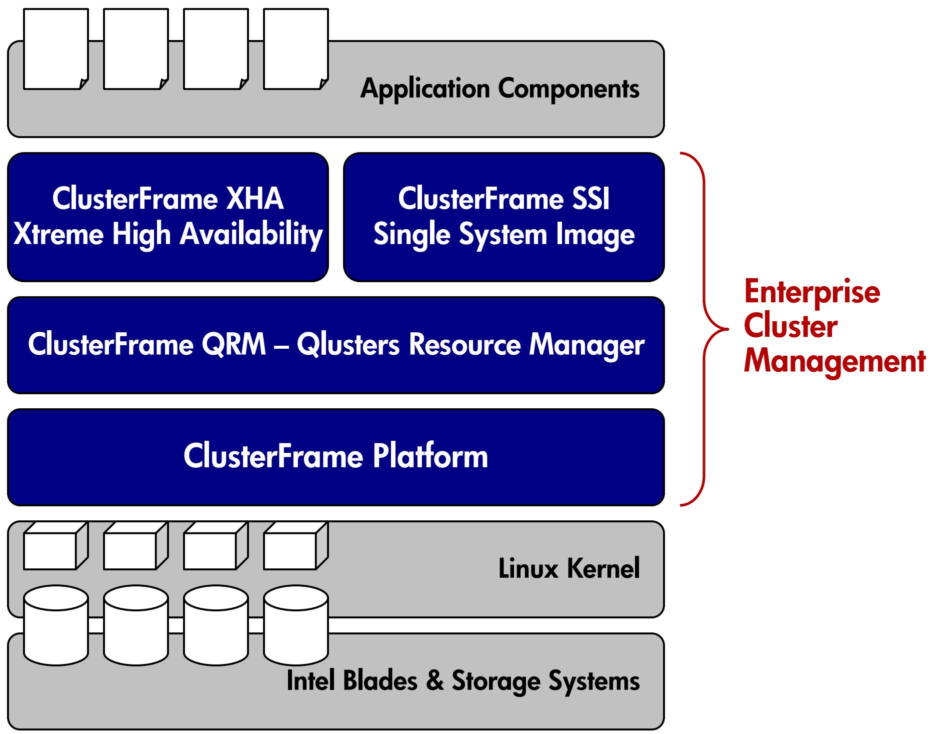
- Completely symmetric Cluster File System with DLM ( no master / slave relationships)
- Each node must be directly attached to SAN
- Limited SSI for management
- No SSI for processes
- No load balancing



# Qlusters

ClusterFrame:

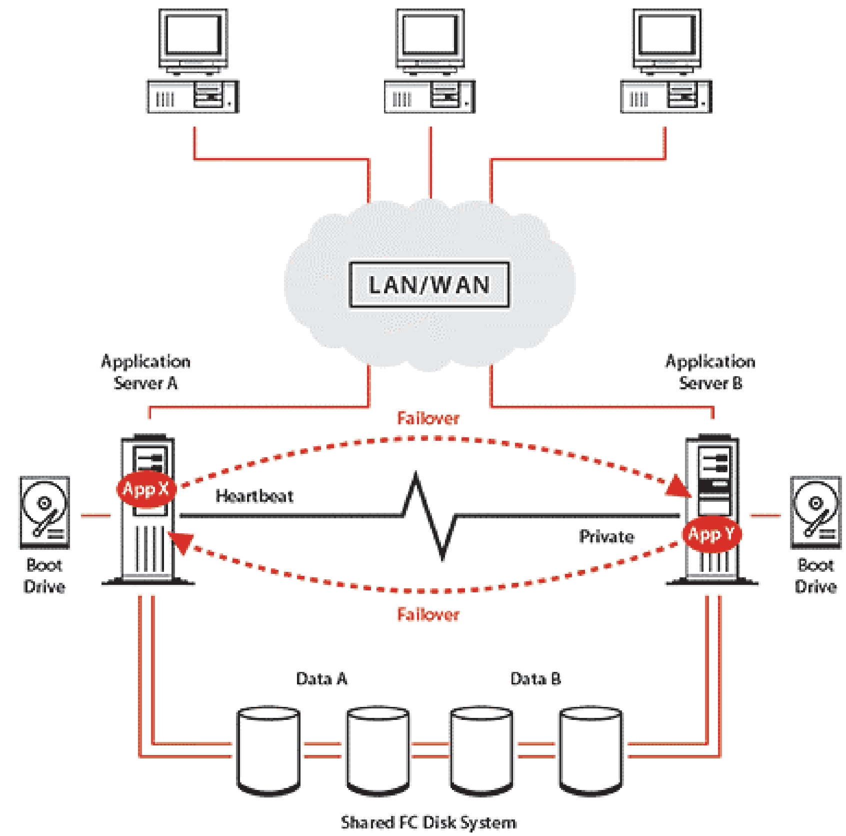
- Based on Mosix
- Uses Home-node SSI
- centralized policy-based management
  - reduces overhead
  - pre-determined resource allowances
  - centralized provisioning
- stateful application recovery



# RedHat GFS – Global File System

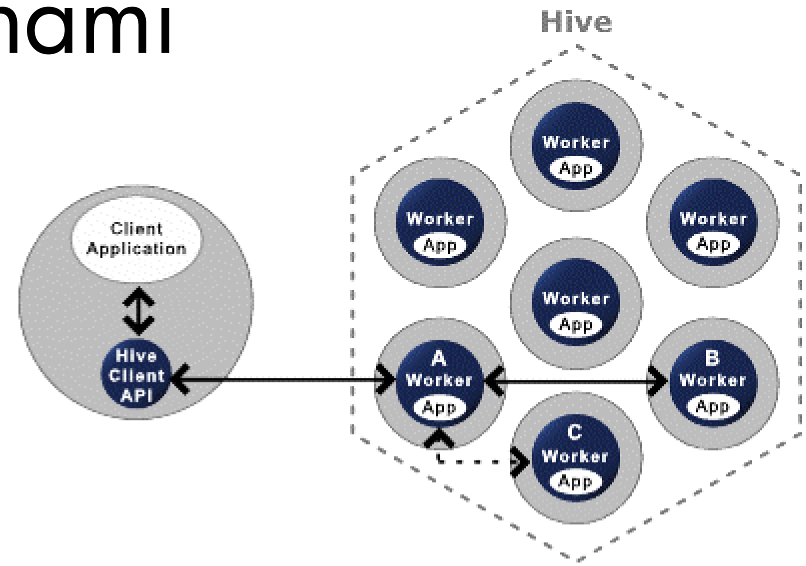
RedHat Cluster Suite (GFS):

- Formerly Sistina
- Primarily Parallel Physical file system (only real form of SSI)
- Used in conjunction with RedHat cluster manager to provide
  - High availability
  - IP load balancing
- Limited sharing and no process load balancing



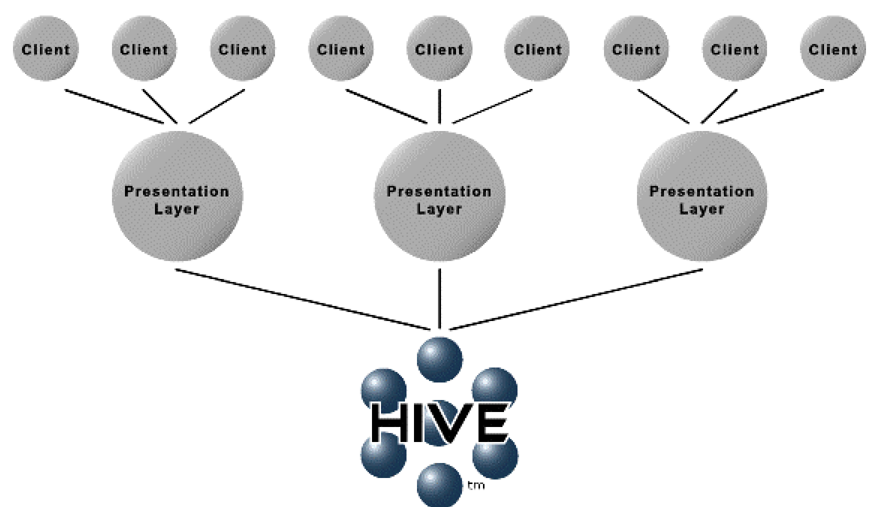

Both servers have redundant connections to disk system(s), but Red Hat Linux Cluster Manager controls access. One server talks to each partition at a time

# Hive Computing - Tsunami




## Hive Creator:

- Hives can be made up of any number of IA32 machines
- Hives consist of:
  - Client applications
  - Hive client API
  - Workers
  - Worker applications
- Databases exist outside of the Hive
- Applications must be modified to run in a Hive
- No Cluster File System
- Closer to Grid model than SSI




HP WORLD 2004
Solutions and Technology Conference & Expo

# Are there Opportunity Gaps in the current SSI offerings?

YES!!

A Full SSI solution is the foundation for simultaneously addressing all the issues in all the cluster solution areas

Opportunity to combine:

- High Availability
- IP load balancing
- IP failover
- Process load balancing
- Cluster filesystem
- Distributed Lock Manager
- Single namespace
- Much more …

# What is a Full Single System Image Solution?

Complete Cluster looks like a single system to:

- Users;
- Administrators;
- Programs and Programmers;

Co-operating OS Kernels providing transparent access to all OS resources cluster-wide, using a single namespace

- A.K.A – You don't really know it's a cluster!

**The state of cluster nirvana**

# SMP – Symmetrical Multi Processing functionality

| Function | SMP |
| --- | --- |
| Manageability | Yes |
| Usability | Yes |
| Sharing / Utilization | Yes |
| High Availability | |
| Scalability | |
| Incremental Growth | |
| Price / Performance | |

# Value add of HA clustering to SMP

| Function | SMP | Traditional Clusters |
|---|---|---|
| Manageability | Yes | |
| Usability | Yes | |
| Sharing / Utilization | Yes | |
| High Availability | | Yes |
| Scalability | | Yes |
| Incremental Growth | | Yes |
| Price / Performance | | Yes |

# SSI Clusters have the best of both!!

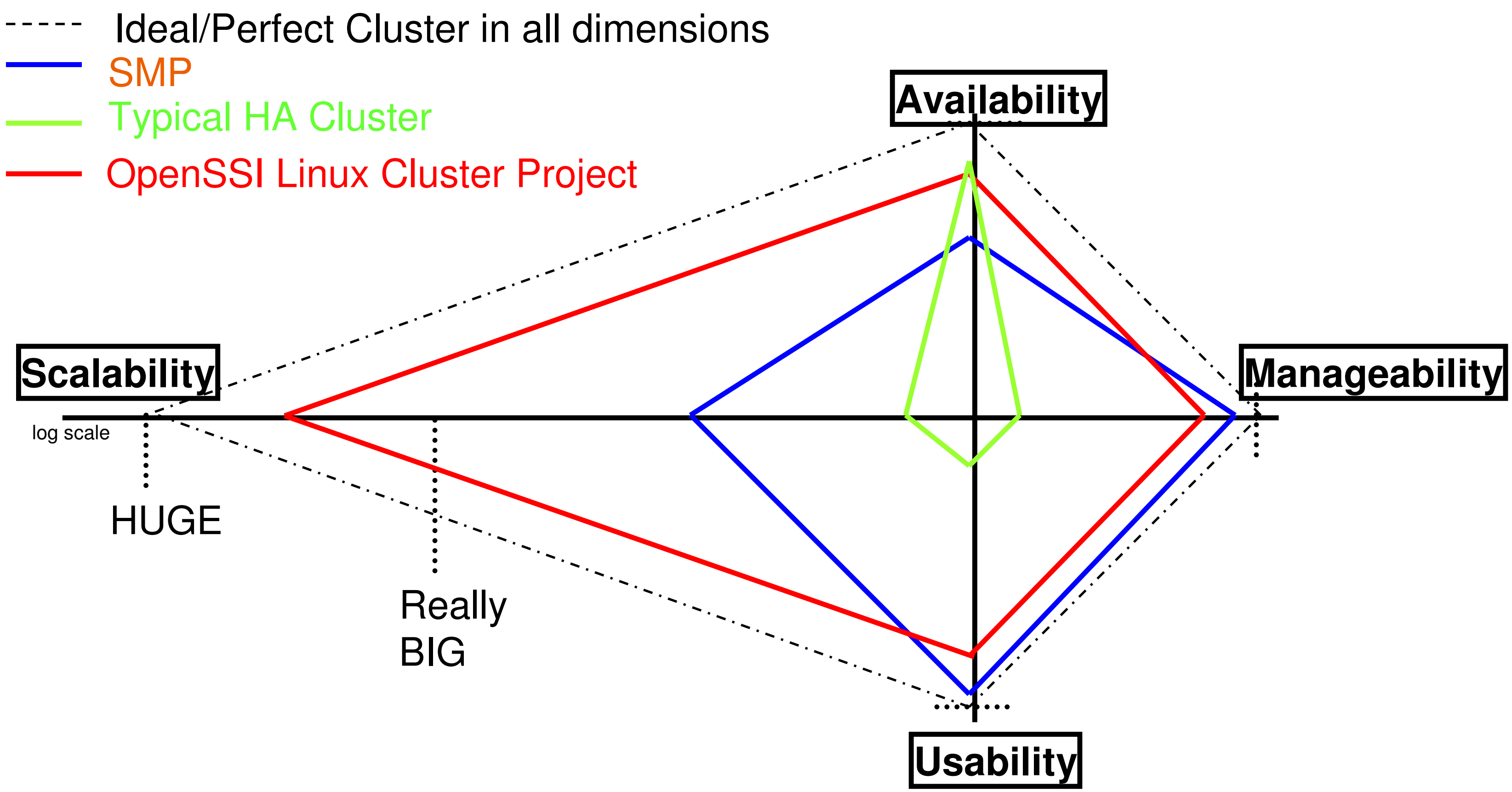
| Function | SMP | Traditional Clusters | SSI Clusters |
|---|---|---|---|
| Manageability | Yes | | Yes |
| Usability | Yes | | Yes |
| Sharing / Utilization | Yes | | Yes |
| High Availability | | Yes | Yes |
| Scalability | | Yes | Yes |
| Incremental Growth | | Yes | Yes |
| Price / Performance | | Yes | Yes |

# Common Clustering Goals

One or All of:

- High Availability
  - A compute engine is always available to run my workload
- Scalability
  - As I need more resource I can access it transparently to the end user application
- Manageability
  - I can guarantee some level of service because I can efficiently monitor, operate and service my compute resources
- Usability
  - Compute resources are assembled together in such a way as to give me trouble free easy operations of my compute resources without regard to having knowledge of the cluster

# OpenSSI Linux Cluster Project



# Overview of OpenSSI Cluster

- Single HA root filesystem
- Consistent OS kernel on each node
- Cluster formation early in boot
- Strong Membership
- Single, clusterwide view of files, filesystems, devices, processes and ipc objects
- Single management domain
- Load balancing of connections and processes

# OpenSSI Cluster Project Availability

- No Single (or even multiple) Point(s) of Failure
- Automatic Failover/restart of services in the event of hardware or software failure
- Application Availability is simpler in an SSI Cluster environment;  statefull restart easily done;
- SSI Cluster provides a simpler operator and programming environment
- Online software upgrade
- Architected to avoid scheduled downtime

# OpenSSI Cluster Project
# Price / Performance Scalability

- What is Scalability?
  - Environmental Scalability and Application Scalability!
- Environmental (Cluster) Scalability:
  - more USEABLE processors, memory, I/O, etc.
  - SSI makes these added resources useable

# OpenSSI Cluster Project
# Price / Performance Scalability

## Application Scalability:

- SSI makes distributing function very easy
- SSI allows sharing of resources between processes on different nodes (all resources transparently visible from all nodes):
  - filesystems, IPC, processes, devices*, networking*
- SSI allows replicated instances to co-ordinate (almost as easy as replicated instances on an SMP; in some ways much better)
- Load balancing of connections and processes
- OS version in local memory on each node
- Industry Standard Hardware (can mix hardware)
- Distributed OS algorithms written to scale to hundreds of nodes (and successful demonstrated to 133 blades and 27 Itanium SMP nodes)

# OpenSSI Linux Cluster - Manageability

- Single Installation
  - Joining the cluster is automatic as part of booting and doesn't have to managed
- Trivial online addition of new nodes
- Use standard single node tools (SSI Administration)
- Visibility of all resources of all nodes from any node
  - Applications, utilities, programmers, users and administrators often needn't be aware of the SSI Cluster
- Simpler HA (High Availability) management

# OpenSSI Linux Cluster
# Single System Administration

- Single set of User accounts (not NIS)
- Single set of filesystems (no “Network mounts”)
- Single set of devices
- Single view of networking
- Single set of Services (printing, dumps, networking*, etc.)
- Single root filesystem (lots of admin files there)
- Single set of paging/swap spaces (goal)
- Single install
- Single boot and single copy of kernel
- Single machine management tools

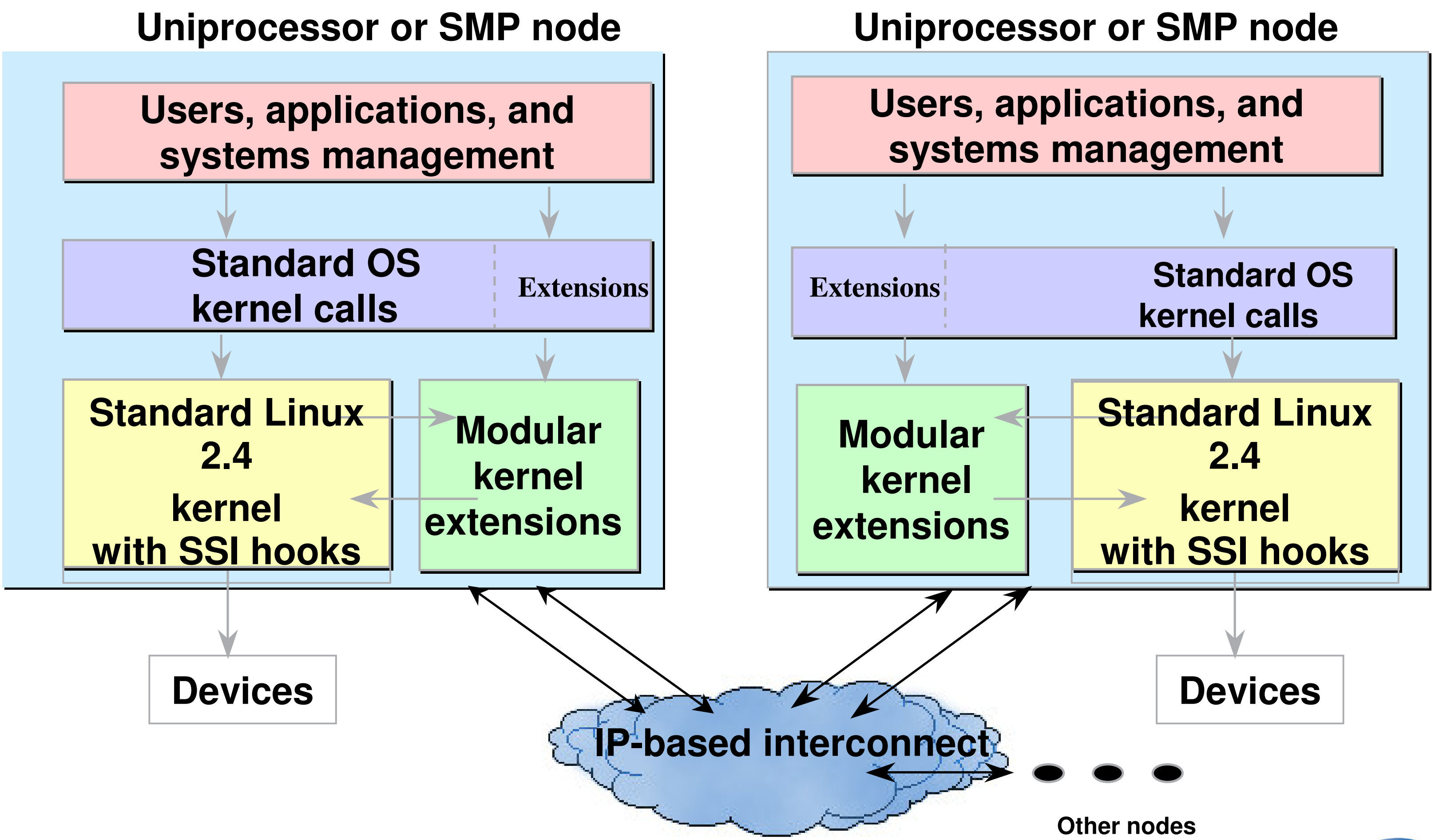
# OpenSSI Linux Cluster - Ease of Use

- Can run anything anywhere with no setup;
- Can see everything from any node;
- Service failover/restart is trivial;
- Automatic or manual load balancing;
  - powerful environment for application service provisioning, monitoring and re-arranging as needed

# Blades and OpenSSI Clusters

- Very simple provisioning of hardware, system and applications
- No root filesystem per node
- Single install of the system and single application install
- Nodes can netboot
- Local disk only needed for swap but can be shared
- Blades don't need FCAL connect but can use it
- Single, highly available IP address for the cluster
- Single system update and single application update
- Sharing of filesystems, devices, processes, IPC that other blade "SSI" systems don't have
- Application failover very rapid and very simple
- Can easily have multiple clusters and then trivially move nodes between the clusters

# How Does SSI Clustering Work?



# Components of Strict SSI

- Single File hierarchy
- Single I/O space
- Single Process Management
- Single Virtual Networking
- Single IPC space and access
  - pipes, semaphores, shared memory, sockets, etc.
- Single system management and user interface
- Single Memory Space *******

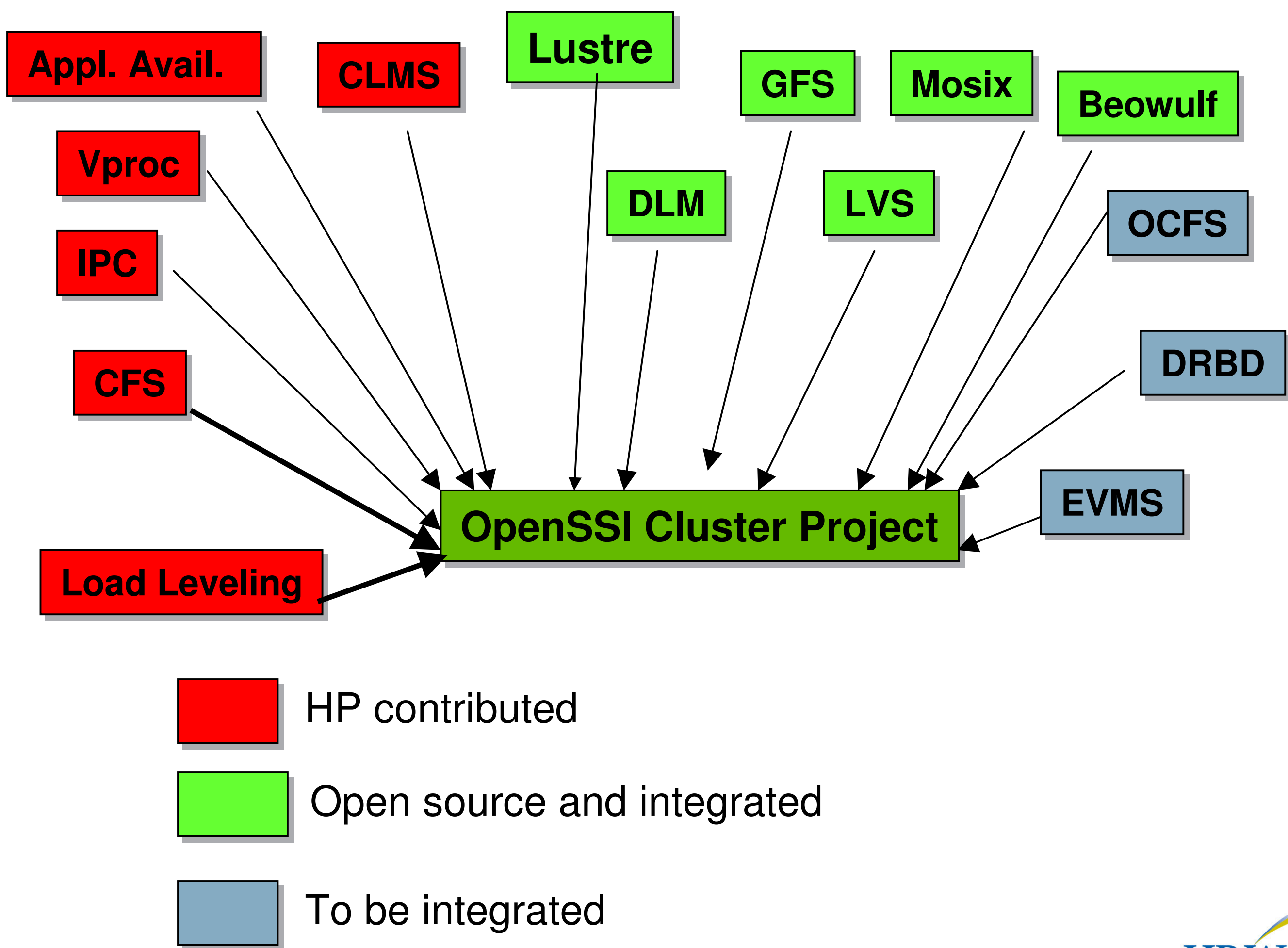
# Added Components for SSI+

- Cluster Membership (CLMS) and membership APIs
- Internode Communication Subsystem (ICS)
- Cluster Volume Manager
- Distributed Lock Manager
- Process migration/re-scheduling to other nodes
- Load-leveling of processes and internet connections
- Single simple installation
- High Availability Features (see next slide)

# Added HA Components for SSI+

- Basically anything pure HA clusters have:
  - device failover and filesystem failover
  - HA interconnect
  - HA IP address or addresses
  - Process/Application monitoring and restart
- Transparent filesystem failover or parallel filesystem

# Component Contributions to OpenSSI Cluster Project




HP WORLD 2004 Solutions and Technology Conference & Expo

# Component Contributions to OpenSSI Cluster Project

- **LVS - Linux Virtual Server**:
  - front end director (software) load levels connections to backend servers
  - can use NAT, tunneling or redirection
    - (we are using redirection)
  - can failover director
  - integrated with CLMS but doesn't use ICS
  - http://www.LinuxVirtualServer.org

# Component Contributions to OpenSSI Cluster Project

- **GFS, openGFS:**
  - parallel physical filesystem; direct access to shared device from all nodes;
  - Sistina has proprietary version (GFS) (now RH has it)
    - http://www.sistina.com/products_gfs.htm
  - project was using open version (openGFS)
    - http://sourceforge.net/projects/opengfs

# Component Contributions to OpenSSI

- **Lustre:**
  - open source project, funded by HP, Intel and US National Labs
  - parallel network filesystem;
  - file service split between a metadata service (directories and file information) and data service (spread across many data servers (stripping, etc.)
  - operations can be done and cached at the client if there is no contention
  - designed to scale to thousands of clients and hundreds of server nodes
    - http://www.lustre.org

# Component Contributions to OpenSSI Cluster Project

- **DLM - Distributed Lock Manager:**
  - IBM open source project (being taken over)
  - Is now used by openGFS
  - http://sourceforge.net/projects/opendlm

# Component Contributions to OpenSSI Cluster Project

- **DRBD - Distributed Replicated Block Device**:
  - open source project to provide block device mirroring across nodes in a cluster
  - can provide HA storage made available via CFS
  - Works with OpenSSI
  - http://drbd.cubit.at

# Component Contributions to OpenSSI Cluster Project

- **Beowulf :**
  - MPICH and other Beowulf subsystems just work on OpenSSI
    - Ganglia, Scalable PBS, Maui, ….

# Component Contributions to OpenSSI Cluster Project

- **EVMS - Enterprise Volume Management System**
- not yet clusterized or integrated with SSI
- **http://sourceforge.net/projects/evms/**

# SSI Cluster Architecture/ Components

13. Packaging and Install

14. Init;booting; run levels

15. Sysadmin;

18.Timesync (NTP)

16. Appl Availability; HA daemons

17. Application Service Provisioning

19. MPI, etc.

Kernel Interface

1. *Membership*

3. Filesystem
- CFS
- Physical filesystems
- GFS
- Lustre

5. Process Loadleveling

4. Process Mgmt

6. IPC

7. Networking/ LVS

8. DLM

9. Devices/ shared storage devfs

10. Kernel Replication Service

2. *Internode Communication/ HA interconnect*

11. CLVM/ EVMS (TBD)

12. DRBD (TBD)

HP WORLD 2004
Solutions and Technology Conference & Expo

# OpenSSI Linux Clusters - Status

- Version 1.0 just released –
  - Binary, Source and CVS options
  - Functionally complete RH9 and RHel3
  - Debian release also available
  - IA-32 and Itanium Platforms
  - Runs HPTC apps as well as Oracle RAC
  - Available at openssi.org

# OpenSSI Linux Clusters - Conclusions

- HP has recognized that Linux clusters are important part of the future.
- HP has recognized that combining scalability with availability and manageability/ease-of-use is key to clustering
- HP is leveraging its merger with Compaq (Tandem/Digital) to bring the very best of clustering to a Linux base

# Backup material

# 1. SSI Cluster Membership (CLMS)

- CLMS kernel service on all nodes
- Current CLMS Master on one node
- Cold SSI Cluster Boot selects master (can fail to another node)
  - other nodes join in automatically and early in kernel initialization
- Nodedown detection subsystem monitors connectivity
  - rapidly inform CLMS of failure (can get sub-second detection)
  - excluded nodes immediately reboot (some integration with STOMITH still needed)
- There are APIs for membership and transitions

# 1. Cluster Membership APIs

- cluster_ name()
- cluster_membership()
- clusternode_num()
- cluster_transition()  and cluster_detailedtransition()
  - membership transition events
- clusternode_info()
- clusternode_setinfo()
- clusternode_avail()

- Plus command versions for shell programming

# 2. Inter-Node Communication (ICS)

- Kernel to kernel transport subsystem
  - runs over TCP/IP
  - Structured to run over VI or other messaging systems
- RPC, request/response, messaging
  - server threads, queuing, channels, priority, throttling, connection mgmt, nodedown, ...
- Bonding for HA interconnect

# 3. Filesystem Strategy

- Support parallel physical filesystems (like GFS), layered CFS (which allows SSI cluster coherent access to non-parallel physical filesystems (JFS, XFS, reiserfs, ext3, cdfs, etc.) and parallel distributed (eg. Lustre)
- transparently ensure all nodes see the same mount tree

# 3. Cluster Filesystem (CFS)

- Single root filesystem mounted on one node
- Other nodes join root node and “discover” root filesystem
- Other mounts done as in std Linux
- Standard physical filesystems (ext2, ext3, XFS, ..)
- CFS layered on top (all access thru CFS)
  - provides coherency, single site semantics, distribution and failure tolerance
- transparent filesystem failover

# 3. Filesystem Failover for CFS - Overview

- Dual or multi-ported Disk strategy
- Simultaneous access to the disk not required
- CFS layered/stacked on standard physical filesystem and optionally Volume mgmt
- For each filesystem, only one node directly runs the physical filesystem code and accesses the disk until movement or failure
- With hardware support, not limited to only dual porting
- Can move active filesystems for load balancing
- Processes on client nodes see no failures, even if server fails (transparent failover to another server)

# 3. Filesystem Model – GFS/OpenGFS

- Parallel physical filesystem; direct access to shared device from all nodes;
- Sistina has proprietary version (GFS)
  - http://www.sistina.com/products_gfs.htm
- Project is currently using open version (openGFS)
  - http://sourceforge.net/projects/opengfs

# 3. Filesystem Model - Lustre

- Open source project, funded by HP, Intel and US National Labs
- Parallel network filesystem;
- File service split between a metadata service (directories and file information) and data service (spread across many data servers (stripping, etc.)
- Operations can be done and cached at the client if there is no contention
- Designed to scale to thousands of clients and hundreds of server nodes
  - http://www.lustre.org

# 4. Process Management

- Single pid space but allocate locally
- Transparent access to all processes on all nodes
- Processes can migrate during execution (next instruction is on a different node; consider it rescheduling on another node)
- Migration is via servicing /proc/<pid>/goto (done transparently by kernel) or migrate syscall (migrate yourself)
- Also rfork and rexec syscall interfaces and onnode and fastnode commands
- process part of /proc is systemwide (so ps & debuggers “just work” systemwide)
- Implemented via a virtual process (Vproc) architecture

# 4. Vproc Features

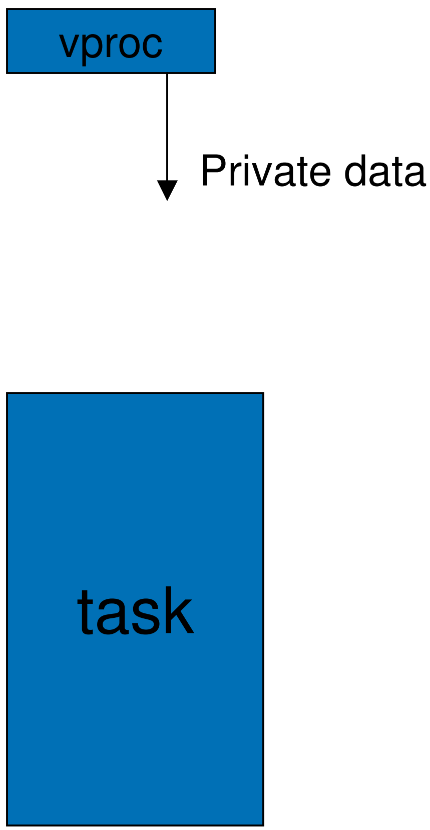
- Process always executes system calls locally
- No “do-do” at “home node”; never more than 1 task struct per process
- For HA and performance, processes can completely move
  - Therefore can service node without application interruption
- Process always only has 1 process id
- Clusterwide job control
- Architecture to allow competing remote process implementations

# 4. Process Relationships

- Parent/child can be distributed
- Process Group can be distributed
- Session can be distributed
- Foreground pgrp can be distributed
- Debugger/ Debuggee can be distributed
- Signaler and process to be signaled can be distributed
- All are rebuilt as appropriate on arbitrary failure
- Signals are delivered reliably under arbitrary failure

# Vproc Architecture - Data Structures and Code Flow

Data structures




Code Flow

Base OS code calls vproc interface routines for a give vproc

Define interface

Replaceable vproc code handles relationships and sends messages as needed; calls pproc routines to manipulate task struct; may have it’s own private data

Define interface

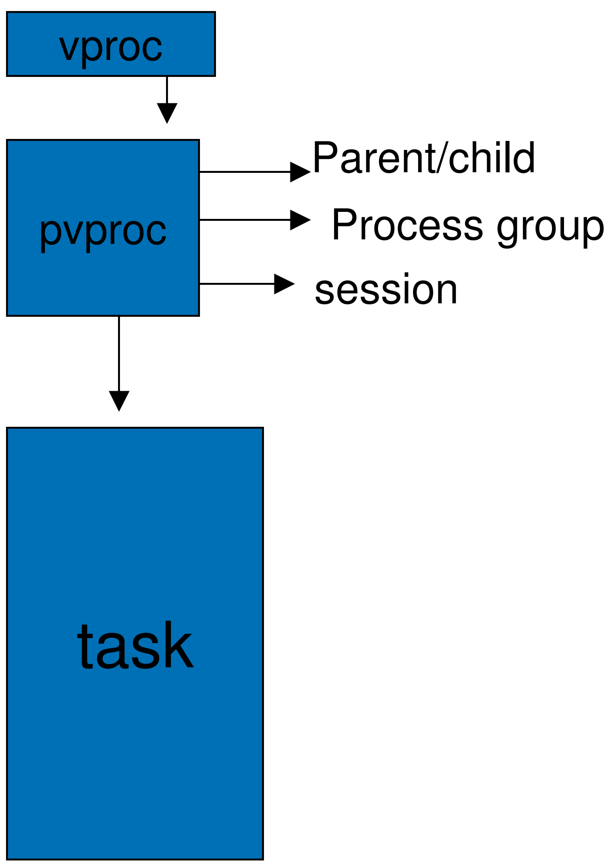
Base OS code manipulates task structure

# 4. Vproc Implementation

- Task structure split into 3 pieces:
  - vproc (tiny, just pid and pointer to private data)
  - pvproc (primarily relationship lists; …)
  - task structure
- all 3 on process execution node;
- vproc/pvproc structs can exists on other nodes, primarily as a result of process relationships

# Vproc Implementation - Data Structures and Code Flow

Data structures




Code Flow

Base OS code calls vproc interface routines for a give vproc

Define interface

---

Replaceable vproc code handles relationships and sends messages as needed; calls pproc routines to manipulate task struct

Define interface

---

Base OS code manipulates task structure

# Vproc Implementation - Vproc Interfaces

- High level vproc interfaces exist for any operation (mostly system calls) which may act on a process other than the caller, or may impact a process relationship.   Examples are sigproc, sigpgrp, exit,  fork relationships, …
- To minimize “hooks” there are no vproc interfaces for operations which are done strictly to yourself (eg. Setting signal masks)
- Low level interfaces (pproc routines) are called  by vproc routines for any manipulation of the task structure

# Vproc Implementation - Tracking

- Origin node (creation node; node whose number is in the pid) is responsible for knowing if the process exists and where it is execution (so there is a vproc/pvproc struct on this node and a field in the pvproc indicates the execution node of the process); if a process wants to move, it must only tell it's origin node;
- If the origin node goes away, part of the nodedown recovery will populate the "surrogate origin node", whose identity is well known to all nodes; never a window where anyone might think the process did not exist;
- When the origin node reappears, it resumes the tracking (lots of bad things would happen if you didn't do this, like confusing others and duplicate pids)
- If the surrogate origin node dies, nodedown recovery repopulates the takeover surrogate origin;

# Vproc Implementation - Relationships

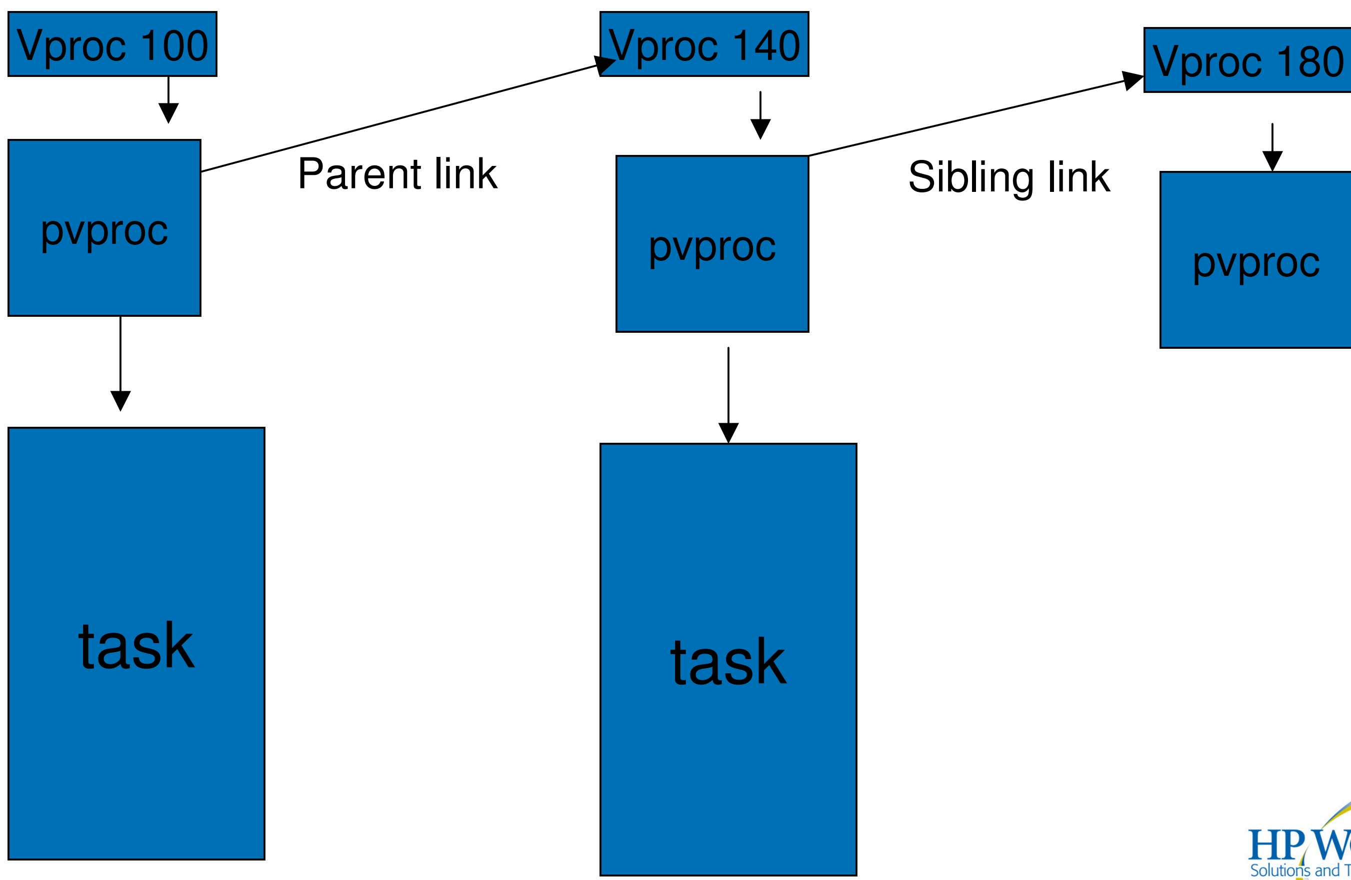
- Relationships are handled through the pvproc struct and not task struct;
- Relationship list (linked list of vproc/pvproc structs) is kept with the list leader (e.g.. Execution node of the parent or pgrp leader)
- Relationship list sometimes has to be rebuilt due to failure of the leader (e.g.. Process groups do not go away when the leader dies)
- Complete failure handling is quite complicated - published paper on how we do it.

# Vproc Implementation - parent/child relationship

Parent process (100) at it's execution node

Child process 140 running at parent's execution node

Child process 180 running remote



# Vproc Implementation - APIs

- rexec()- semantically identical to exec but with node number arg - can also take “fastnode” argument
- rfork()- semantically identical to fork but with node number arg
  - can also take “fastnode” argument
- migrate() - move me to node indicated; can do fastnode as well
- /proc/<pid>/goto causes process migration
- where_pid() - way to ask on which node a process is executing

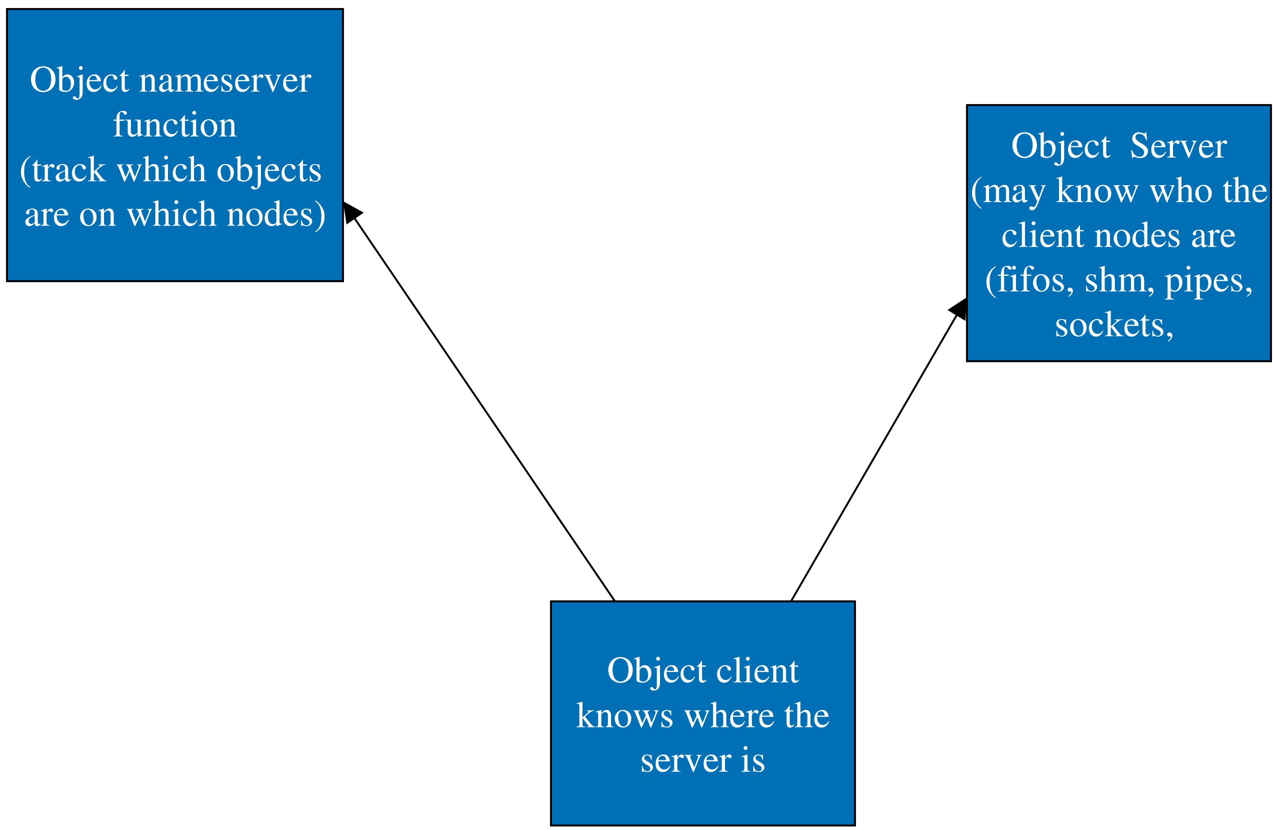
# 5. Process Load Leveling

- There are two types of load leveling - connection load leveling and process load leveling
- Process load leveling can be done “manually” or via daemons (manual is onnode and fastnode;  automatic is optional)
- Share load info with other nodes
- each local daemon can decide to move work to another node
- Adapted from the Mosix project load-leveling

# 6. Interprocess Communication (IPC)

- Semaphores, message queues and shared memory are created and managed on the node of the process that created them
- Namespace managed by IPC nameserver (rebuilt automatically on nameserver node failure)
- pipes and fifos and ptys and sockets are created and managed on the node of the process that created them
- All IPC objects have a systemwide namespace and accessibility from all nodes

# 6. Basic IPC model



# 7. Networking/LVS - Linux Virtual Server

- Front end director (software) load levels connections to backend servers
- Can use NAT, tunneling or redirection
  - (we are using redirection)
- Can failover director
- Integrated with CLMS but doesn't use ICS
- Some enhancements to ease management
- http://www.LinuxVirtualServer.org

# 8. DLM - Distributed Lock Manager

- IBM open source project (abandoned and saved)
- hopefully it will be used by openGFS
- http://sourceforge.net/projects/opendlm

# 9. Systemwide Device Naming and Access

- Each node creates a device space thru devfs and mounts it in /cluster/nodenum#/dev
- Naming done through a stacked CFS
- each node sees it's devices in /dev
- Access through remote device fileops (distribution and coherency)
- Multiported can route thru one node or direct from all
- Remote ioctls can use transparent remote copyin/out
- Device Drivers usually don't require change or recompile

# 11. EVMS/CLVM

## EVMS - Enterprise Volume Management System

- not yet clusterized or integrated with SSI
- http://sourceforge.net/projects/evms/

## CLVM - Cluster Logical Volume Manager

- being done by Sistina (not going to be open source)
- not yet integrated with SSI
- http://www.sistina.comp/products_lvm.htm

# 12. DRBD - Distributed Replicated Block Device

- open source project to provide block device mirroring across nodes in a cluster
- can provide HA storage made available via CFS
- not yet integrated with SSI
- http://drbd.cubit.at

# 13. Packaging and Installation

- First Node:
  - install standard distribution
  - Run SSI RPM and reboot SSI kernel
- Other Nodes:
  - can PXE/netboot up and then use shared root
  - basically a trivial install (addnode command)

# 14. Init, booting and Run Levels

- Single init process that can failover if the node it is on fails
- nodes can netboot into the cluster or have a local disk boot image
- all nodes in the cluster run at the same run level
- if local boot image is old, automatic update and reboot to new image

# 15. Single System Administration

- Single set of User accounts (not NIS)
- Single set of filesystems (no “Network mounts”)
- Single set of devices
- Single view of networking (with multiple devices)
- Single set of Services (printing, dumps, networking*, etc.)
- Single root filesystem (lots of admin files there)
- Single install
- Single boot and single copy of kernel
- Single machine management tools

# 16. Application Availability

- “Keepalive” and “Spawndaemon” part of base NonStop Clusters technology
- Provides User-level application restart for registered processes
- Restart on death of process or node
- Can register processes (or groups) at system startup or anytime
- Registered processes started with “Spawndaemon”
- Can unregister at any time
- Used by the system to watch daemons
- Could use other standard application availability technology (eg. Failsafe or ServiceGuard)

# 16. Application Availability

- Simpler than other Application Availability solutions
  - one set of configuration files
  - any process can run on any node
  - Restart does not require hierarchy of resources (system does resource failover)
  - Can use std “services” management for automatic failover/restart

# 19. Beowulf

- MPICH libraries and mpirun just work in the SSI cluster
- LAMPI has also been adapted
- Job launch, job monitoring and job abort much simpler in OpenSSI environment
- http://www.beowulf.org

# OpenSSI Linux Clusters - Conclusions

- OpenSSI is an attempt to provide a common cluster framework for all forms of clustering
- OpenSSI simultaneous addresses availability, scalability, manageability and usability
- Lots of neat stuff all together in the OpenSSI project
- Demonstrated on 25-node Blade system
- Tested to 132 nodes using Proliant rack systems