The Case for A Hierarchal System Model for Linux Clusters

Mark Seager and Brent Gorda,
Lawrence Livermore National Laboratory
April 6, 2009
Version 2

Motivation

The computer industry today is no longer driven, as it was in the 40s, 50s and 60s, by High-performance computing requirements. Rather, HPC systems, especially Leadership class systems, sit on top of a pyramid investment mode. Figure 1 shows a representative pyramid investment model for systems hardware. At the base of the pyramid is the huge investment (order 10s of Billions of US Dollars per year) in semiconductor fabrication and process technologies. These costs, which are approximately doubling with every generation, are funded from investments multiple markets: enterprise, desktops, games, embedded and specialized devices. Over and above these base technology investments are investments for critical technology elements such as microprocessor, chipsets and memory ASIC components. Investments for these components are spread across the same markets as the base semiconductor processes investments. These second tier investments are approximately half the size of the lower level of the pyramid. The next technology investment layer up, tier 3, is more focused on scalable computing systems such as those needed for HPC and other markets. These tier 3 technology elements include networking (SAN, WAN and LAN), interconnects and large scalable SMP designs. Above these is tier 4 are relatively small investments necessary to build very large, scalable systems high-end or Leadership class systems. Primary among these are the specialized network designs of vertically integrated systems, etc.

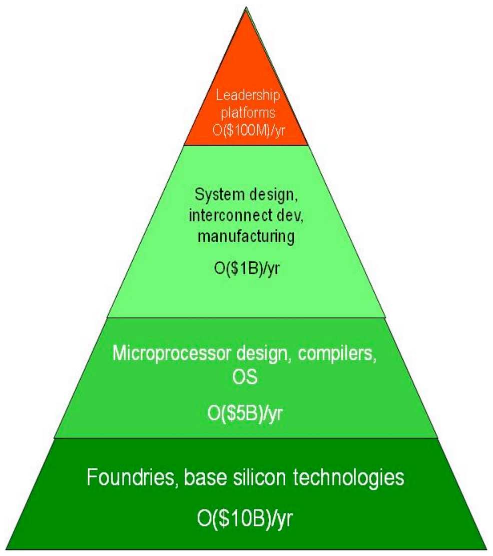
Leadership platforms O($100M)/yr
System design, interconnect dev, manufacturing
O($1B)/yr
Microprocessor design, compilers, OS
O($5B)/yr
Foundries, base silicon technologies
O($10B)/yr

Figure 1: Leadership-class HPC systems sit on top of a $15B+ pyramid of investment.

The Hierarchal Systems Model of Petascale Systems

Since the mid-1990s Linux clusters and proprietary, vertically integrated systems (PVIS) have leveraged the above hardware and software pyramid investment model. The gap between the scalability of COTS Linux clusters and PVIS systems have diminished in the intervening years and now form a major fraction of the TOP500 list. However, with recent development in PVIS, such as IBM BlueGene and Cray XT4, the scalability of PVIS has again vastly outstripped basic Linux clusters. By looking at lessons learned in the march to petascale PVIS, we have learned that one must focus on three things: **scalability** of hardware, **scalability** of system software and infrastructure and applications **scalability**. Key observations on hardware and system software scalability coming out of the BlueGene experience are: 1) keep the highly replicated hardware and software components as simple possible and still get the job done (known as KISS, or Keep It System, Stupid); 2) applying a "factor and simplify" design methodology[1] leads to a hierarchal system model for both hardware and software; 3) the runtime environment (including the OS and system services) felt by applications must extremely low noise. These design principles lead to an extremely simple (small parts count) compute node implementation with MTBF measured in the field of about 3 millennia. On the software side the highly replicated unit is the light weight kernel (LWK). Due to the simplicity of the compute node architecture all external (but not interconnect) I/O and other OS functionality is function shipped to IO nodes (ION) with an external SAN interface. This creates a hierarchal system model where there is a large number of CN and a reasonable number of ION (about the same size as a small to medium size Linux cluster). If we now add a few Login nodes where users login interact with the system (e.g., code development, batch job management and visualization) and a few Service Nodes for the RAS infrastructure and scalable system administration, then we have the basis for a fully hierarchal system infrastructure. For example, job launch and debugger daemons can be migrated off the compute nodes (and thereby reduce the system noise and improve software reliability by keeping the CN LWK environment simple as possible) to the ION.

[1] The "factor and simplify" design methodology takes a seemingly impossible problem (e.g., scaling Linux OS to 65,536 way parallelism for BlueGene/L) and breaks it into two problems; one of which is easy to solve and the other is merely difficult (e.g., a light weight kernel on the 65,536 compute nodes (the difficult piece) and function ship to Linux on 1,024 IO Nodes (the solved piece)).

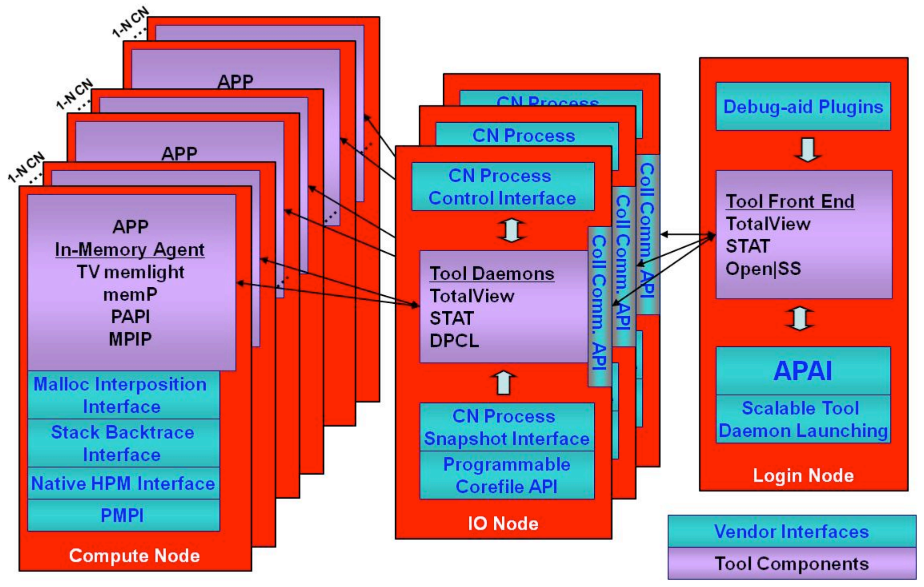

Figure 2: Hierarchal PVIS system model showing how the next generation of scalable tools can naturally use this hierarchal hardware and software infrastructure.

This offers the opportunity for a hierarchal system infrastructure that associates a set of ION with CN at job launch time and job launch is also hierarchal in the sense that a user submits the job and the batch system, running on a SN launches job steps (parallel jobs) that start by launching daemons on behalf of the user on the ION and then those daemons launch the job (and later manipulate it for the debugger and other performance analysis tools) onto the CN LWK. This “factor and simplify” approach also serendipitously provides a fan out infrastructure for tools and other system services. This fan out infrastructure approach provides unique opportunities for scalability. For example, debuggers when setting memory watch points or conditional memory watch points require processing every time each MPI task touches a page in memory containing the target memory address as most implementations use page table (or similar) mechanism to trap memory references with little performance impact on memory operations on watched pages. However upon this hardware page trap, the debugger must then determine if the memory address referenced in the page is the one being monitored or not and check to see if the condition is met, if there is one. This processing typically is today serialized back to the debugger process running on the Login node and interacting with the user. This is not scalable. With the hierarchal infrastructure, the debugger daemon running on each of the ION can process all of the page faults from MPI tasks on the CN under its dominion. This process runs in parallel across all the ION. As the job grows, so does the number of ION associated with it and the method describe is thereby scalable.

The Scalability Dilemma for Exascale Systems Has at Least Two Horns

Although significant research needs to be done on system scalability for Exascale systems, it is clear that a hierarchal system model, possibly with multiple levels in the hierarchy, is at least an intermediate step or starting point for research activities. The second horn of the Exascale systems scalability dilemma is that if PVIS systems drift too far away from where Linux clusters are, then the pyramid investment model in Figure 1 breaks down. It breaks down because more and more specialized technology will have to be developed for the PVIS and less and less leverage is obtained from lower levels in the pyramid. Thus we need to keep Linux clusters scaling up to petascale and beyond as we push PVIS systems technology to the Exascale. Thus the march to Exascale must be two pronged: scale PVIS to Exascale and COTS Linux clusters to petascale and beyond.

Current Flat Linux Cluster System Model

To understand the gaps here for Linux clusters it is instructive to review the current state of the art in Linux cluster design and deployment methodology.

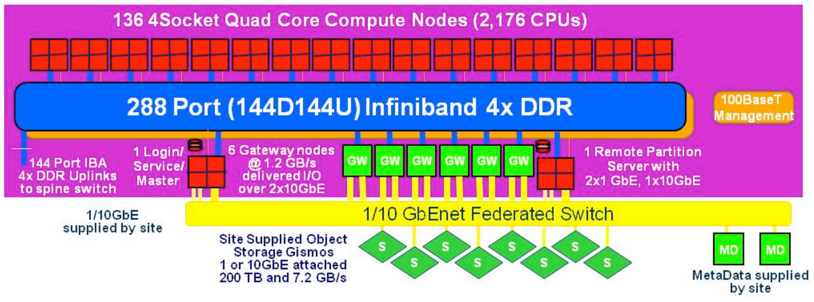
136 4Socket Quad Core Compute Nodes (2,176 CPUs)
288 Port (144D144U) Infiniband 4x DDR
100BaseT Management
144 Port IBA 4x DDR Uplinks to spine switch
1 Login/ Service/ Master
6 Gateway nodes @ 1.2 GB/s delivered I/O over 2x10GbE
GW
GW
GW
GW
GW
GW
1 Remote Partition Server with 2x1 GbE, 1x10GbE
1/10GbE supplied by site
1/10 GbEnet Federated Switch
Site Supplied Object Storage Gismos 1 or 10GbE attached 200 TB and 7.2 GB/s
S
MD
MD
MetaData supplied by site

Figure 3: Linux Clusters of various sizes can be economically built from a Scalable Unit concept.

Recent advances in design indicate that multiple Linux clusters can be more economically built, integrated and operated by adopting a Scalable Unit (SU) design methodology. These and other Linux cluster designs in common use today essentially present a "flat system" model. SU are small aggregates of nodes that contain all the essential elements and node types necessary to build Linux clusters of various sizes: even vary large ones. In Figure 3, a SU design based on the 288 port IBA 4x DDR switch is depicted. The preponderance of nodes are CN as these are where the user MPI based applications run. The remaining nodes perform systems (and Login) functions and hence are kept minimal. In this SU design, we have the minimum of Login Nodes (LN at 1) and Remote Partition Servers (RPS at 1) and a few gateway nodes (GW at 4) necessary to provide sufficient IO bandwidth for applications running on the cluster over a SAN to the Lustre (or other) global (accessible multiple Linux clusters), parallel (supporting parallel IO within a cluster) file system. When building Linux clusters of various sizes the system functions also grow linearly and scale appropriately. For example, the RPS remote boots all the diskless nodes in the cluster (CN and GW) and serve up root and swap partitions for each node. Since this function is replicated independently in each SU these services scale with system size. For large clusters all one needs to add is a way to configure multiple RPS nodes in parallel from a single management workstation attached to the cluster over a management Ethernet.

Proposed Model

From the above discussion, we notice that a slight tweak on the Linux cluster "flat system model" based on SU design point can yield a hierarchal system model and offer the potential to scale Linux clusters to 10K-100K nodes. It turns out, from the hardware side, only a slight shift is necessary:

1. Design and build compute node as simple as possible (KISS)
2. Use gateway nodes as ION
3. Use RPS nodes as cluster of service nodes

The with recent advances in microprocessor design (e.g., including memory controllers and memory buses directly on the processor) and the tendency of the industry to aggregate more function onto the processor with time, it is possible to envision a very simple node design and a path to get there quickly.

On the software side a moderate shift is necessary to bridge the gap:

1. Light weight (low noise) Kernel
2. Function shipping interface to ION
3. All system services off of ION, only minimal job launch on CN
4. Debugging and process manipulation interface on ION to CN processes
5. Distributed RAS DB and infrastructure

Filling this gap will require significant effort by the Linux community. However, there has been a lot of research and development out of DOE SciDAC (e.g., FASTOS effort) that can be harvested. In addition, many vendors have indicated a willingness to commercialize such a model for the community.

This would be a good example of how we can change the industry by utilizing the R&D➔D&E➔Commercialization mechanisms described in a companion white paper titled "A Collaboration and Commercialization Model for Exascale Software Research."

MAIN PRINCIPLES

1. HPC pyramid investment model requires we pull up the rest of the pyramid while pushing to exascale or the model breaks down.
2. Hierarchal systems model developed for petascale systems is a good starting point, with possibly more than one level in the hierarchy, for exascale systems research
3. The current "Flat" Linux cluster systems model can be turned into a hierarchal systems model and scale up to 10K to 100K nodes.
4. A change to both hardware (simpler compute nodes) and software are required.
5. We can mine existing petascale systems efforts and combine it with readily available commercialization paths.