Abstract

Since 2001 we have used FreeBSD as a high performance computing (HPC) cluster operating system. In the process we have ported a number of HPC tools including Ganglia, Globus, Open MPI, and Sun Grid Engine. In this talk we will discuss the process of porting these types of applications and issues encountered while maintaining these tools. In addition to generally issues of porting code from one Unix-like operating system to another, there are several type of porting common to many HPC infrastructure codes which we will explore. Beyond porting, we will discuss how the ports collection aids our use of HPC applications and ways we think overall integration could be improved.

Cluster Computing At The Aerospace Corporation

The Fellowship Cluster

(Photo/The Aerospace Corporation)

- 352 dual processor nodes running FreeBSD
- Gigabit networking

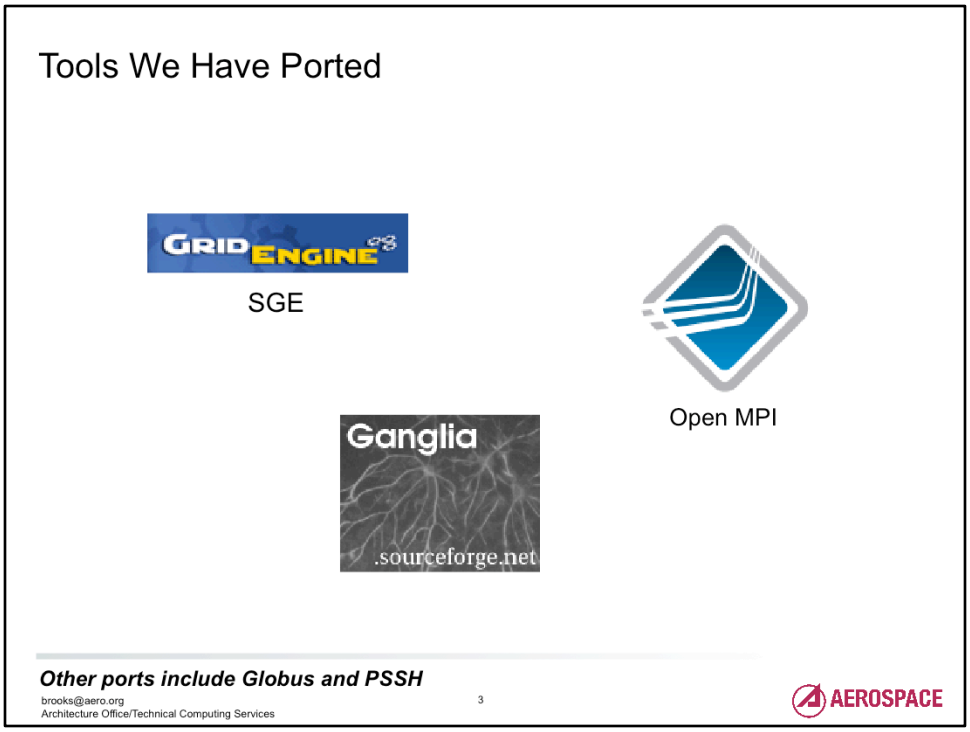
Our largest HPC resource

brooks@aero.org
Architecture Office/Technical Computing Services

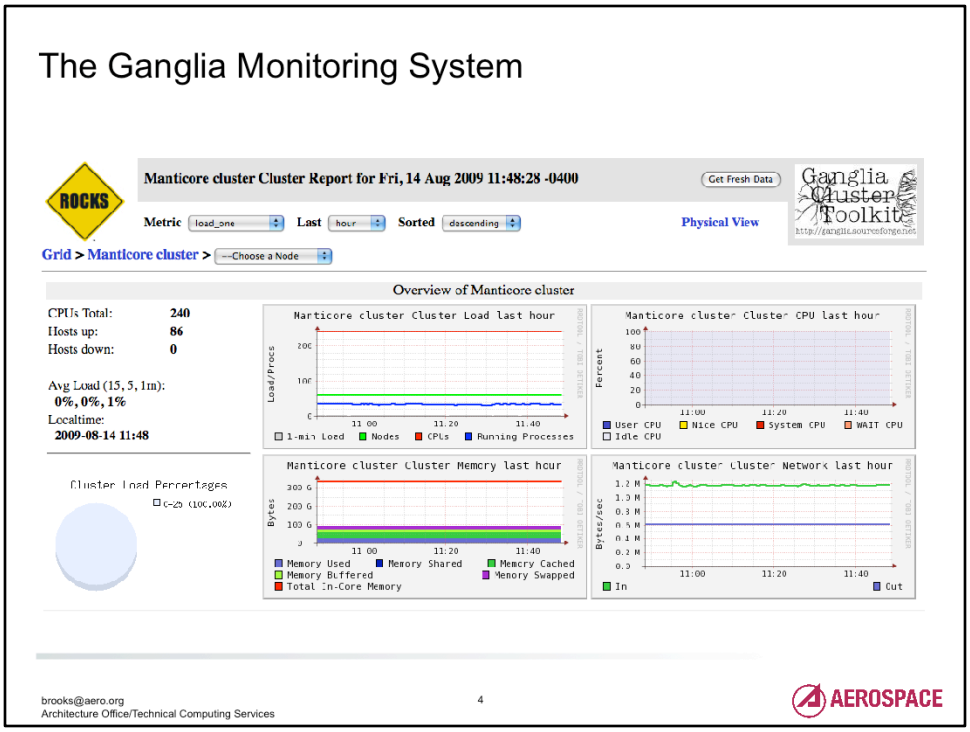
2

AEROSPACE

At Aerospace we have designed, built, and operated a FreeBSD HPC cluster since 2001. This picture shows the Fellowship cluster in it's current form with 352 dual processor nodes. In the process of building and running Fellowship we have ported a number of open source HPC tools to FreeBSD. In this talk I will discuss the process of porting them, issues we encountered, and a few pet peeves about application portability.



Some of the tools we have ported to FreeBSD include Sun Grid Engine—also known as SGE—a batch job manager; the Ganglia monitoring system, a cluster/grid monitoring tool; and Open MPI a leading implementation of the Message Passing Interfaces which is a toolkit for message based parallel programming.



The Ganglia monitoring systems, usually refered to as Ganglia, is a cluster and grid monitoring tool that provides current and historical data on the status of a cluster or collection of clusters. The data includes CPU count, CPU utilization, memory use, and network I/O. Ganglia is used on systems ranging from small clusters to PlanetLab, a globally distributed network of over 1000 nodes at nearly 500 sites.

http://www.planet-lab.org/

Ganglia Metric Structure

- Daemons on each node send metrics to a central collector
- Meterics cover system configuration and load
- Metrics may be fixed or variable
- Examples:
  - *Operating system*
  - *CPU count and type*
  - *CPU utilization breakdown*
  - *System memory allocation*

- Providers are functions that return string or numeric values
- Previous versions hard code a table of function pointers
  - *More recent versions have pluggable metrics*

brooks@aero.org
Architecture Office/Technical Computing Services

Ganglia consists of daemons on each node send periodic updates of various metrics to a central collector that stores the results in a set of RRD databases. The metrics cover many aspects of system configuration and load and may either be fixed (per-boot) or vary with time. Example metrics include operating system, CPU count and type, cpu utilization breakdown, and system memory allocation.

Metric providers are functions the return string or numeric values. In early versions of Ganglia the set of metrics was determined at compile time. In more recent versions, it is configured at runtime and pluggable metric providers are supported.

Porting Ganglia Metrics: General Process

- Examine the code for other operating systems to determine what the metric really means
- Find a way to get the desired value from normal user tools
  - *Look at top(1), ps(1), procstat(1), sysctl(1), etc*
- Take advantage of open source and read the code to the utilities in question
- Read man pages on the functions discovered
- Either write appropriate code or (if the licenses allow) appropriate it from utility in question

brooks@aero.org
Architecture Office/Technical Computing Services

AEROSPACE

The process I used to port metrics is pretty simple. First, I checked the code for other operating systems to determine what the metric means. Then I figured out a way to use standard tools such as top, ps, procstat, or sysctl to extract that information from the system. Once appropriate tools were identified, I examined their source code to determine how they retrieved the information in question and used that information along with man pages to write an appropriate metric implementation. In many cases it was possible to copy the code directly from FreeBSD utilities along with another copyright statement and license block because Ganglia is BSD licensed.

Ganglia Metric: cpu_num

```
g_val_t
cpu_num_func ( void )
{
   g_val_t val;
   int ncpu;
   size_t len = sizeof (int);
   if (sysctlbyname("hw.ncpu", &ncpu, &len, NULL, 0)
     == -1 || !len)
        ncpu = 1;

   val.uint16 = ncpu;
   return val;
}
```

brooks@aero.org
Architecture Office/Technical Computing Services

AEROSPACE

One of the simplest metrics that requires operating system interaction (as opposed to looking at the ganglia config files) is the cpu_num metric. The cpu_num metric shows the number of cpus in the system. As such it's about the easiest function to write. We simply observed that the easiest way to get the number of cpus on FreeBSD was the hw.ncpus sysctl and wrote code to obtain that value.

XXX: add animation to point out the uint16 bit?

Ganglia Metrics: swap_total, swap_free

- Administrators obtain swap information from swapinfo(1)

```
$ swapinfo
Device          1K-blocks     Used    Avail Capacity
/dev/ad4s1b       2011128        0  2011128     0%
```

- The swapinfo(1) command (actually usr/sbin/pstat) obtains swapinfo either via the Kernel Memory library (libkvm) or by sysctl.
- We support both in Ganglia
  - *libkvm: Supports older versions of FreeBSD*
  - *sysctl: Does not require root (dropped after startup)*
  - *Selection at runtime based on sysctl MIB presence*

brooks@aero.org
Architecture Office/Technical Computing Services

AEROSPACE

The swap_total and swap_free metrics represent several interesting cases including the use of the kvm(3) interface and supporting multiple interfaces for cross platform support. In porting them we observed that users and administrators can obtain swap information from the swapinfo(1) command. After determining that it was actually a link to pstat, we dug into the code and determined that it used libkvm to access the data if called on a coredump and a sysctl if not. When we first ported Ganglia, FreeBSD 4 did not support the sysctl interface so we supported both because the sysctl interface allowed us to run ganglia without ever being root, but the libkvm interface as needed since we used FreeBSD 4.x. As I will show, our implementation probes at runtime and prefers the sysctl interface.

Ganglia Metrics: swap_total, swap_free (cont)

Probing for sysctl support

```
g_val_t metric_init(void)
{
   g_val_t val;

   mibswap_size = MIB_SWAPINFO_SIZE;
   if (sysctlnametomib("vm.swap_info", mibswap, &mibswap_size) == -1) {
      kd = kvm_open(NULL, NULL, NULL, O_RDONLY, "metric_init()");
   } else {
      kd = kvm_open(_PATH_DEVNULL, NULL, NULL, O_RDONLY, "metric_init()");
      use_vm_swap_info = 1;
   }
   pagesize = getpagesize();
   <...>
   val.int32 = SYNAPSE_SUCCESS;
   return val;
}
```

brooks@aero.org
Architecture Office/Technical Computing Services

AEROSPACE

Here we have a partially redacted version of the FreeBSD metric_init() function. It shows the portion of the function which checks for the availability of the vm.swap_info MIB and then either opens and stores a descriptor pointing to the running kernel's memory or stores a descriptor pointing to /dev/null to allow sysctl based libkvm commands to work. It also caches the system page size and the fact that the swap_info MIB was found.

Ganglia Metrics: swap_total, swap_free (cont)

The core of swap_total_func()

```
if (use_vm_swap_info) {
     for (n = 0; ; ++n) {
       mibswap[mibswap_size] = n;
       size = sizeof(xsw);
       if (sysctl(mibswap, mibswap_size + 1, &xsw, &size, NULL, 0) == -1)
          break;
       if (xsw.xsw_version != XSWDEV_VERSION)
          return val;
        totswap += xsw.xsw_nblks;
      }
  } else if(kd != NULL) {
     n = kvm_getswapinfo(kd, swap, 1, 0);
     if (n < 0 || swap[0].ksw_total == 0)
        val.f = 0;
     totswap = swap[0].ksw_total;
   }
  val.f = totswap * (pagesize / 1024);
```


Here we see the core of swap_total_func(). Depending on the results of metric_init(), the amount of swap used is either retrieved using sysctl() or libkvm. In the sysctl case, we retrieve information from each individual swap device and total them where the libkvm interface provides a direct total. There are a couple other interesting things of note here. First, ganglia expects swap size to be a floating point number of KiB. Variants of this are common in ganglia due to the desire to represent large sizes identically on all machines. Another interesting thing is the XSWDEV_VERSION line. This points out a problem with version numbers in binary interfaces. They are all well and good, but clients need to know about the new version so in practice they do little to help provide ABI stability.

Ganglia Metrics: Memory Use

- A difficult set of metrics to implement
- Ganglia memory types: total, buffers, cached, free, shared
- Memory types in top(1): active, inactive, wired, cache, buffers, free
- Possible values for total: hw.physmem, hw.realmen, hw.usermem, total of values from top(1)?

Ganglia Metric	Source In Port
mem_total	hw.physmem
mem_buffers	vfs.bufspace
mem_cached	vm.stats.vm.v_cache_count
mem_free	vm.stats.vm.v_free_count
mem_shared	0

Free memory is wasted memory. –anonymous

brooks@aero.org
Architecture Office/Technical Computing Services

AEROSPACE

The ganglia metrics for memory use were some of the hardest to port and some of the one's I'm least satisfied with overall. The problem stems from the fact that the different virtual memory systems in different operating systems use vastly different sets of accounting buckets. The Ganglia metrics total, buffers, cached, free, and shared appear to be derived from values that are easy to obtain on Linux. Unfortunately, they don't match any of the major sets of memory use FreeBSD outputs. For example top(1) shows active, inactive, wired, cache, buffers, and free memory where free memory is always a small number of a system that has been up and active for some time since the VM sees little need to waste CPU time freeing memory that might be used again. As the saying goes free memory is wasted memory in the operating system. Even total memory is complicated. For example we have both hw.physmem the actual amount of memory in the system (modulo 32-bit limits on 32-bit, non-PAE system) and hw.realmem, the memory that it is possible to use. The table shows the set of mappings we chose, but this is far from optimal.

Aside: Schema Issues

- Scheme mismatches are a persistent problem
- Memory use
- TCP, UDP, NFS, etc statistics
  - *Different systems count or trace different events*
- "Local" or "real" disk usage
  - *Which FS types are local?*

- More flexible metric designs can help
  - *Hierarchical metrics?*
- Build in appropriate metadata to avoid hard coding
  - *Add bits to FS info? Are file systems on a SAN local?*

brooks@aero.org
Architecture Office/Technical Computing Services

Related to the memory metric issues, there is a general class of issues with scheme mismatches. Others include things like network protocol statistics where different implementations may present much more or less information or present the same information is different ways. Another common issue is determining the disk space used or available on disks and wanting to distinguish between local, remote, and pseudo file systems. Programs tend to end up with hard coded lists of local file systems. This scales poorly and fails to answer questions like, "Are file systems on a SAN local?"

Aside: IPv6 Socket Behavior

- "By default, FreeBSD does not route IPv4 traffic to AF_INET6 sockets. The default behavior intentionally violates RFC2553 for security reasons. Listen to two sockets if you want to accept both IPv4 and IPv6 traffic." – FreeBSD Kernel Interfaces Manual: inet6(4)
- IPv6 support in Ganglia resulting in breaking IPv4 support on FreeBSD
- The socket code was actually in the Apache Portable Runtime
  - *It has been fixed and references to this issue appear even in userspace*
- After much discussion the code was refractored to support multiple sockets.
- This continues to present a significant portability barrier.

brooks@aero.org
Architecture Office/Technical Computing Services

AEROSPACE

One of the more annoying porting issues we enountered was with IPv6 support. KAME derived IPv6 stacks tend to violate RFC 2553 and disallow IPv4 mapped IPv6 sockets from receiving IPv4 data and instead require two sockets to be opened. When ganglia introduced IPv6 support they did so using the Apache Portable Runtime (apr) which did not understand this issue. As a result, IPv6 support disabled IPv6 support on FreeBSD. After much wrangling, ganglia was modified to open two sockets as required under FreeBSD. Given that code tend's to expect this behavior to work, I remain unconvinced that this incompatibility it worth it.

Sun Grid Engine

A Leading Open Source Batch Scheduler

- Batch job scheduler and resource manager
  - *Also handles interactive jobs*
- One of the leading schedulers
- Purchased by Sun and released under an open source license in July 2001
- Ron Chen started a FreeBSD port and Aerospace completed it

brooks@aero.org
Architecture Office/Technical Computing Services

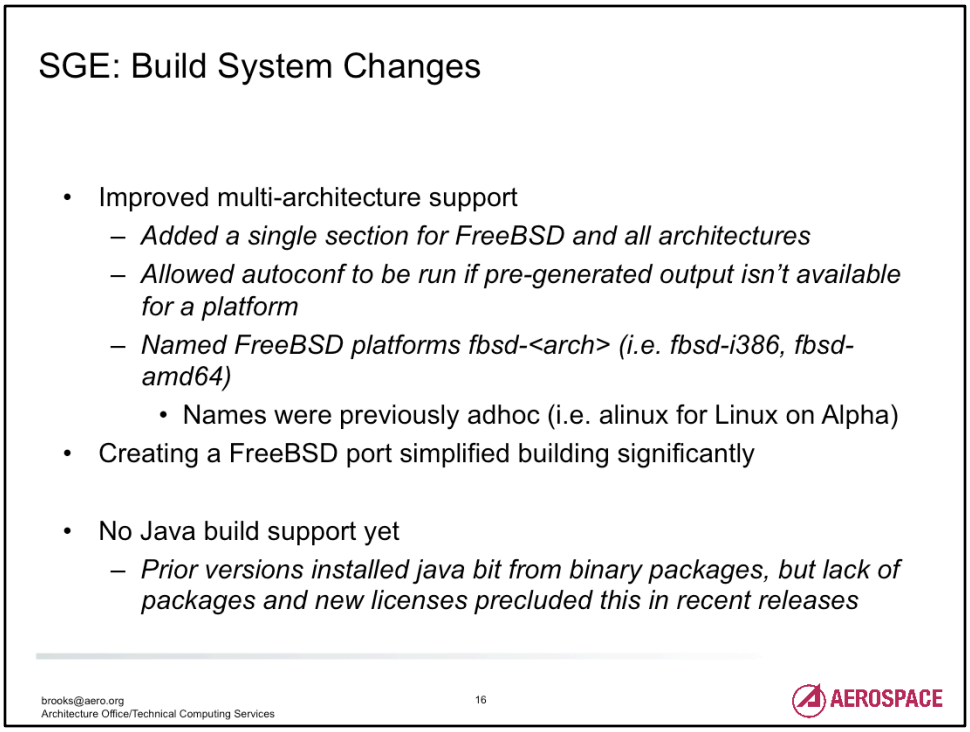
AEROSPACE

Sun Grid Engine is one of the leading open source batch schedulers and resource managers. The other credible option is Torque, a fork of OpenPBS. Prior to Sun's purchase of Grid Engine and the subsequent open source release, we had been attempting to get OpenPBS to work in our environment, but it was never stable for us (or many others on the OpenPBS mailing lists). When SGE was release and Ron Chen started a port we leapt at the chance to try something else and started a port our selves.

SGE: Build System Overview

- 2900-plus line csh script called aimk
  - *No relation to the pvm aimk*
  - *Each OS/architecture had separate configuration sections*
- Custom dependency generator from X11
- Makefiles
- Autoconf for 3rd party packages (qmake, qtcsh)
  - *Mostly pre-generated config.h, Makefiles, etc*
- Apache Ant for Java components

The first step in porting SGE was figuring out the build system. It's a unique and complex system consisting of a nearly three-thousand line csh script confusingly named aimk. Within the script, each pair of operating system and architecture has a separate configuration section. The script invokes a dependency generator, a variety of make instances, autoconf in some cases and in recent versions Apache Ant. Historically autoconf output is prebuilt for each platform so autoconf was not actually run by aimk.



In the process of porting the build system we introduced a number of innovations to reduce the complexity of adding new platforms. Most of these were in the form of adding support for multi-platform builds since we knew from the start we wanted to support at least FreeBSD i386 and amd64. We wanted to be able to support new architectures with little or no additional change to the build system. To the end we added a single configuration section for FreeBSD architectures and changed the architecture naming so that FreeBSD's platform string is always fbsd-<arch> where arch is the machine type as given by uname –m. We also augmented the build system so that for portions of the source tree that depend on autoconf output, we run autoconf if no pregenerated output is available.

Because the build system requires many steps to generate a working system, we also created a FreeBSD port early on so we can build the system repeatably with a reasonable number of command invocations.

One section we have not yet ported is the code to build the Java interfaces. That started as an interface to to the DRMAA job submission interface, but now includes a GUI installer. At one point we were able to harvest Java components from the pre-build binaries, but those are no longer available under an appropriate license so we disable all Java support in the port.

SGE: Portability Defines

- SGE defines a few types and printf format strings for a number of a number of common system definitions
- The type definitions predate C99 fixed with integer types and are very ad hoc
  - *Early versions also supported Cray's unix with 64-bit ints*
- Examples (FreeBSD):

```
#define  u_long32    uint32_t
#define  uid_t_fmt   "%u"
#define  gid_t_fmt   "%u"
#define  pid_t_fmt   "%d"
```

brooks@aero.org
Architecture Office/Technical Computing Services

After the build system, we needed to tackle a variety of portability definitions. Early versions of Grid Engine were ported to systems that significantly predate modern versions of the C standard such as C99. As a result there are several sets of type definitions for fixed width integers as well as a number of definitions to handle differences in handling required to print a number of standard Unix typedefs like uid_t, gid_t, and pid_t. A fair bit of this code could be simplified today, but doing so would probably require removing support for some older systems like Cray and HP-UX.

SGE: Host Metrics

- Host metrics similar to Ganglia
  - *More scattered around the source tree and surrounded with ifdefs*
  - *Greping for a platform ifdef (LINUX) provides a way to find them*

```
$ qhost -h r01n01
HOSTNAME                ARCH         NCPU  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
----------------------------------------------------------------------------------
global                  -               -     -       -       -       -       -
r01n01                  fbsd-i386       8  0.00    3.2G  215.1M    3.0G     0.0
```


Like Ganglia, SGE collects a number of metrics from the execution daemons on each node and uses those results to make job placement decisions. The implementation is not as neatly divided into metric functions, but the basic principles of porting metrics are the same. I found the easiest way to find them was to grep for LINUX in the source tree to find all the things Linux had to implement. The Darwin and NetBSD ports seem to have grepped for FREEBSD.

The example here shows some of the metrics collected on nodes.

SGE: Job/Process Metrics

- Process and job data are tracked though a system called Portable Data Collector (PDC) which emulates much of the Irix job system
  - *Processes in a job are tracked by adding an extra per-job group to the group list of any process*
  - *To find process resource use, the process table is walked and processes with the supplemental group are counted*
  - *A similar trick is used to kill the processes in a job*
- Our current implementation uses libkvm to obtain process lists and access process statistics
  - *A sysctl based version would be desirable*

brooks@aero.org
Architecture Office/Technical Computing Services

AEROSPACE

One place where SGE's metrics differ from Ganglia is that SGE wants to track resource use by all the processes that make up a job. This is done in the Portable Data Collector sub system. On Irix, there is a mechanism for attaching job IDs to different processes and then querying resource use for the process and all its children. This feature does not exist on most other platforms so SGE implements a cleaver hack to achieve a similar effect. The trick is that they allocate an otherwise unused group on each node and then add it to the group list for each process. Since the group list is copied to each child on fork() and ordinary users can not adjust their group list this provides a tag which can be used to detect the processes that make up a job. When SGE wants to determine the cumulative resource use of a job, it walks the process table using libkvm and totals all resource use. Recent changes to FreeBSD (available in 7.3 and 8.0) will allow the use of sysctl to perform this task, but that will not be a portable option for some time.

Aside: Signal Handling

- SGE needs the ability to install persistent signal handlers
- SGE used SysV's sigset() which is in FreeBSD
- BSD's signal(3) supports this which differs from POSIX
- Convinced Sun to switch to sigaction(2)

"To make certain that no one could write an easily portable application, the POSIX committee added yet another signal handling environment which is supposenot d to be a superset of BSD and both System-V environments." – Jim Frost

http://www.frostbytes.com/~jimf/papers/signals/signals.html

Another interesting portability issues was introduced when other SGE developers fixed some bugs by adding persistent signal handlers. Unfortunately, they used the easy to use, but non-portable sigset() function. POSIX defines the sigset() function, but FreeBSD does not implement it. In practice our implementation of signal() is equivalent to sigset(), but POSIX compliant implementations are not. In the end, I was able to persuade them to switch all instances of sigset() or signal() to sigaction() which is well defined, but has the unfortunate characteristic of requiring multiple lines of code to replace each simple signal() or sigset() call. The Jim Frost quote here does a good job of portraying my feelings about the state of signal handling.

Open MPI

- Open MPI implementations the Message Passing Interface
  - *Primary toolkit for building message passing parallel applications*
  - *Provides a rich common API*
- Open MPI is Open Source
  - *Basis for commercially supported toolkits such as Sun HPC Tools*
- One of a few leading MPI implementations
  - *Derived in part from LAM-MPI*

brooks@aero.org
Architecture Office/Technical Computing Services

AEROSPACE

Open MPI is a leading open source implementation of the Message Passing Interface aka MPI. MPI is the primary toolkit for building message passing parallel application. These are applications where multiple process coordinate their computations with messages. MPI hides the details of the underlying network beneath a common API. Open MPI is also the basis of commercial toolkits such as Sun's HPC toolkit.

Open MPI: Porting

- One of the easier ports so far
- Highly modular code base with advanced autoconf scripts
- Built on several platforms at launch
  - *Years of HPC experience prove the need for portability*
- We needed to add support for backtrace() and backtract_symbols() from glibc
  - *The devel/libexecinfo port already provided this*

- A not yet ported:
  - *PLPA: Portable Linux Processor Affinity*
    - Hides ABI differences between different processor affinity interfaces
    - Direct implementation using cpuset() should work on FreeBSD

brooks@aero.org
Architecture Office/Technical Computing Services

AEROSPACE

Open MPI was overall on of the easiest ports we have done so far. The code is highly modular with extensive use of autoconf scripts to detect features. It's clear the development team has taken to heart the historical need for portability in HPC code, especially middleware. The one feature we needed in the initial port was the backtrac() and backtrac_symbols() functions glibc implements. Fortunately the devel/libexecinfo port already provides that functionality so we simply added the necessary autoconf bits to do detect it and we had a working port.

The one piece we know has not yet been ported is the CPU affinity support which relies on a side project of Open MPI called Portable Linux Process Affinity (PLPA). PLPA exists to deal with the fact that at least three different syscall ABIs were shipped by different major Linux vendors before a standard version was imported into the official tree. In FreeBSD we don't have to worry abou this problem so a simple mapping between the cpuset() api and PLPA should be possible.

Aside: Supporting Multiple MPI Implementations

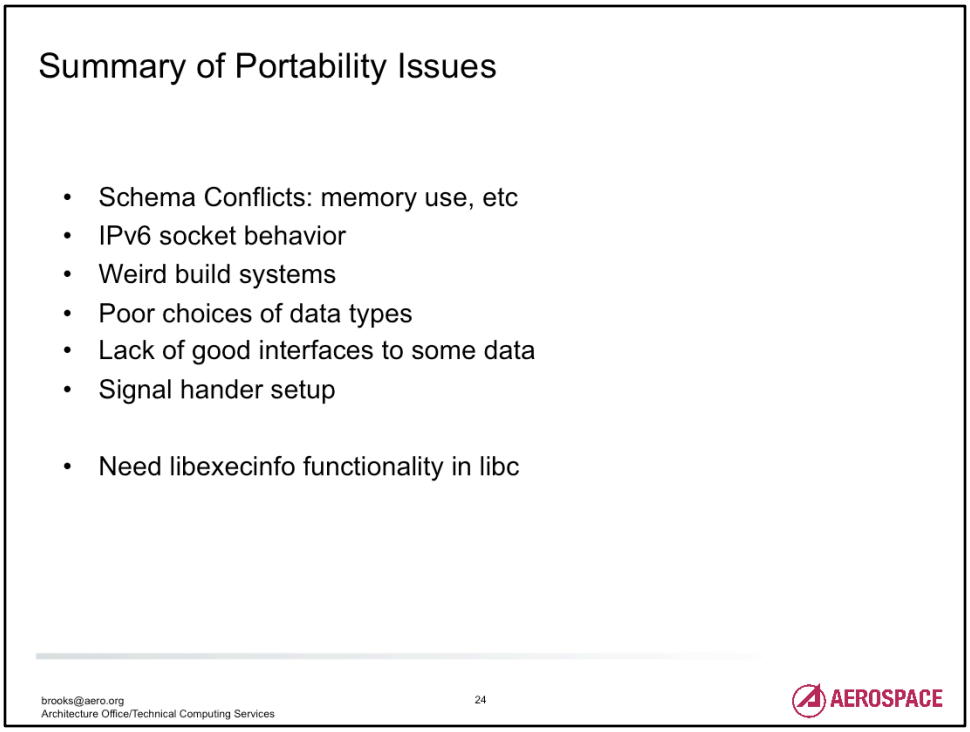
- Open MPI is one of several MPI implementations
- Most sites find it useful to support multiple implementations
- Ports currently install in non-conflicting locations
  - *net/lam - ${PREFIX}*
  - *net/mpich2 - ${PREFIX}/mpich2*
  - *net/openmpi - ${PREFIX}/mpi/mpich2*
- Consistent locations would be better

- MPI implementations provide a series of compiler wrappers to aid in portability of linking
  - *Provide a general wrapper system similar to javavmwrapper?*

brooks@aero.org
Architecture Office/Technical Computing Services

AEROSPACE

In the HPC world it is common to have several implementations of keep tools such as MPI implementations or compilers available so users can choose the best tool for their particular application. With MPI this actually results in a combinatorial explosion as each MPI instance includes compiler wrapper scripts for the particular compiler they were build with. Current FreeBSD ports of MPI implementations each install an a unique location, but the choices are all different. None of them currently handle multiple compiler versions. In my view, it would be ideal if MPI builds installed in consistent locations and multiple compiler versions were supported for the same MPI. Ideally I would also like to see an mpiwrapper similar to the javavmwrapper to allow users to easily switch between MPI versions without too much difficulty.



Having discussed our experiences porting Ganglia, Sun Grid Engine, and Open MPI, I would like to recap the major portability issues I have discussed.

Scheme conflicts for operating system metrics are likely to remain a perpetual problem, but I think there are ways the situation could be improved. The most important of these is it allow schemas to be hierarchical and flexible so that an accurate view can be provided while still allowing general purpose systems to make decisions. In many cases, schemas seem to fall for the trap of generalizing from a single instance and do not rethink often enough.

Another source of compatibility issues is IPv6 socket behavior for IPv4 mapped sockets. BSD IPv6 stacks are non compliant with RFCs. It may be the case that enough software has adapted at this point that changing the standard would be the appropriate response, but I believe individual BSD's should reconsider this KAME decision.

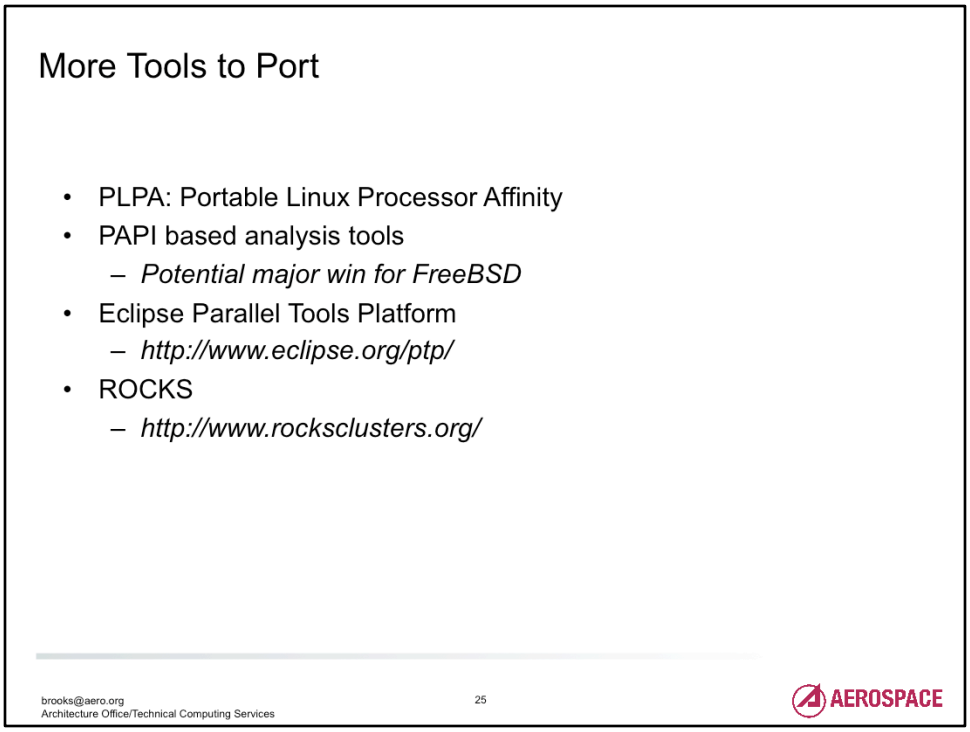
An ongoing issues, especially with long established projects is weird build systems. There isn't really a good general strategy for this problem, but I advise porters to watch out for them and encourage projects to spend the time to reimplement the most strange unconventional ones.

I also discussed the portability issues caused by poor choice of data types. Like weird build systems, this tends to be more common in old code where standards were missing useful features. I encourage projects to purge support for ancient platforms and a decent pace to allow code to take advantage of modern standards.

Another issue I have not talked about explicitly, but which impacts a fair bit of middle ware is lack of good interfaces to access important data. For instance, the need to use a libkvm interface to walk the process table is a sign of insufficiently strong interfaces in the pass. There are now proper interfaces which help resolve these issues, but we are stuck with significant historical baggage. With open source operating systems, porters should consider the possibility of adding new interfaces along with hacking around their lack.

Signal handler setup remains a bit of a problem. The lack of portable APIs other than the highly verbose sigaction() interface seems to lead developers to choose non-portable interfaces. POSIX would do well to revisit this issue and implement an actually simple interface will useful and well defined semantics.

A final issue for FreeBSD in particular is the need for the backtrace*() functions in libc. This just seems like a good idea and we could probably implement it very efficiently using CTF data now that we have DTrace tools in the base system



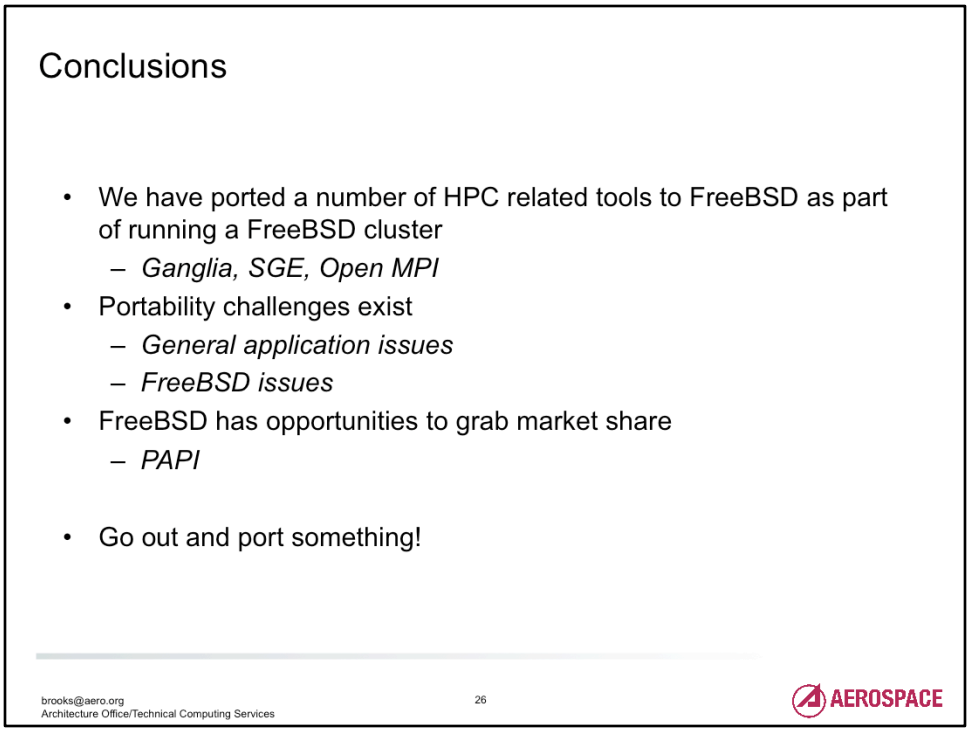
Now that I've talked about porting several HPC tools, I'd like to challenge the audience with some other interesting targets for porting. I've already mentioned PLPA so I will not cover it in more detail here.

An area that may befit FreeBSD greatly is porting the growing number of PAPI applications to FreeBSD. PAPI (http://icl.cs.utk.edu/papi/) attempts to provide a consistent interface to hardware performance counters and it has been ported to FreeBSD HWPMC framework. It is widely used in the HPC community and a number of interesting tools have been built on it. What is interesting from the FreeBSD perspective is that the main line of Linux has not incorporated the patches for performance counters which have been around for over a decade. As a result, only patched kernels can use PAPI. This potentially gives FreeBSD a major advantage since we support HWPMC in our base source tree.

Another interesting tool to port would be the Eclipse Parallel Tools Platform (PTP). PTP aims to provide a complete, open source set of parallel tools. We already support Eclipse, Open MPI, and MPICH2 which are the only dependencies required beyond Eclipse CDT.

A final challenge and by far the most complex is porting FreeBSD to the ROCKS cluster environment. ROCKS is a system for installing and maintaining computing clusters. It works from the basis of an operating system install that can be initiated from PXE and then rides on top of the operating system's package manager to install and configure interesting components. ROCKS was historically based around Red Hat or CentOS Linux verisons, but has been ported to Solaris recently so the worst of the direct Linux and RPM dependencies should be sorted out. I believe a basic port should be fairly straight forward given that FreeBSD is easy to network boot and easy to install manually. Once that's done our packaging system should be entirely adiquate to ROCKS needs.

In general I encourage people to port more HPC tools to FreeBSD. I think the project has a compelling performance story and a great set of tools to support developers.



In conclusion, we have ported a number of tools to FreeBSD. These include Ganglia, SGE, and Open MPI as previously discussed, but also pssh and pypvm. We find that a number of portabitly challenges exist, both in terms of poorly written code and FreeBSD specific issues like IPv4 mapped IPv6 sockets. It seems clear there are areas were FreeBSD can take advantage of ongoing issues in the Linux community. In particular, porting PAPI based tools and the fact that only one cpuset API/ABI exists provide opportunities.

If you want to help FreeBSD succeed, the best advice I can give is to get out there and port something!

Disclaimers

All trademarks, service marks, and trade names are the property of their respective owners.