

---

# Managing System Resources on 'Teras'

Experiences in Managing System Resources  
on a Large SGI Origin 3800 Cluster



The paper to the presentation given at the CUG SUMMIT 2002 held at the University of Manchester.

Manchester (UK), May 20-24 2002.

<http://www.sara.nl>

Mark van de Sanden [Sanden@sara.nl](mailto:Sanden@sara.nl)

Huub Stoffers [hs@sara.nl](mailto:hs@sara.nl)

## Introduction

### Outline of the Paper

This is a paper about our experiences with managing system resources on 'Teras'. Since November 2000 Teras, a large SGI Origin 3800 cluster, is the national supercomputer for the Dutch academic community. Teras is located at and maintained by SARA High Performance Computing (HPC).

The outline of this presentation is as follows:

- Very briefly, we will tell you what SARA is, and what we presently do, particularly in high performance computing, besides maintaining Teras.
- Subsequently, to give you an idea, on the one hand which resources are available on Teras, and on the other hand what resources are demanded, we will give overviews of three particular aspects of the Teras:
  - 1) Some data on the hardware – number of CPUs, memory, etc. - and the cluster configuration, i.e. description of the hosts that constitute the cluster.
  - 2) Identification of the key software components that are used on the system; both, system software components, as well as software toolkits available to users to create their (parallel) applications.
  - 3) A characterization of the 'job mix' that runs on the Teras cluster.
- Thus equipped with a clearer idea of both 'supply' and 'demand' of resources on Teras, we then state the resource allocation policy goals that we pursue in this context, and we review what resource management possibilities we have at hand and what they can accomplish for us, in principle.
- Finally we will go into some of the "gory details" of how we combine tools and control resources in practice. In practice we have stumbled over some shortcomings. There are some problem areas where our control over resource allocation is far less than we want<sup>1</sup>, where we e.g. must refrain from using resource management tools because they have proven not to fit very well with - current implementation of – other crucial software components. We point out these problems and describe the "trade off" choices we made to cope with this state of affairs.

---

<sup>1</sup> And also less than we think is 'reasonable to want'. We are aware of the fact that our policy goals, like all policy goals, have a tendency to formulate an ideal that is never wholly attainable.

## About SARA

SARA was founded in 1971, by the two Amsterdam universities, 'The University of Amsterdam' (UvA) and 'The Free University' (VU), as well as the 'Center for mathematics'. From the outset SARA was a center for high performance computing – measured by the standards of the day of course. Its very raison d'être was to provide and maintain an IT-infrastructure that all three founders needed, but could not afford on their own.

Since then, SARA has extended both its services portfolio and its audience. SARA does other things besides maintaining high performance computer installations and consultancy on vectorization and parallelization. And when we do HPC, its no longer limited to the facilities for the Amsterdam universities. We try to maintain and extend our expertise to different architectures. At present our main HPC installations are:


- IBM SP (Power3 CPUs in 4 CPU 'Winterhawk' nodes, 16 CPU 'Nighthawk' nodes, and Power4 CPUs in 32 CPU 'Regatta' nodes).
- SGI Origin 2000 (MIPS R10000 CPUs) and Origin 3000 (MIPS R14000 CPUs) Series.
- Various Linux clusters, based on Compaq Alpha (1 CPU per node), AMD Athlon (Intel compatible 32bit CPUs, 1 CPU per node) and Intel Itanium (64 bit CPUs, 4 CPUs per node).

In 1990 NCF was founded to centralize on a national level both the financial means as well as strategical decision making concerning high end computing infrastructure for the academic community. SARA has successfully applied for the job of housing and maintaining all national supers thus financed so far: a number of Cray computers in a row, and since November 2000 the SGI Origin 3800, 'Teras', have all been maintained by SARA.

## Overview of the Teras Cluster

### Basic Hardware Specification

In total, the Teras has 1024 MIPS R14000 CPUs, running at a clock speed of 500 MHz., and one Terabyte of memory. The theoretical peak performance of the system as a whole is 1 Teraflop/sec. But Teras is not a monolithic system. Rather, it is a small cluster of large – not to say huge – SGI Origin 3800 multiprocessor distributed shared memory systems. There is 1 GB of memory per CPU. Four CPUs grouped together with 4 GB of **local** memory constitute a single tightly coupled building block – a 'compute brick' or 'C-brick' in SGI terms. The basic Teras hardware specifications and performance and capacity numbers are summarized in the 'slide 5', taken from the Manchester CUG summit presentation, below.



### Basic Teras Hardware Specifications

- 1024 x 500 MHz MIPS R14,000 CPUs
- 1 Teraflops/sec. theoretical peak performance
- 1 Terabyte of memory (1Gigabyte/cpu)
- 10 Terabytes of net on-line RAID5 storage (FC RAID array, 10,000 RPM disks)
- 100 terabytes of off-line storage (tapes for data migration, archiving and backup)

CUG Summit May 20th 2002, Managing Resources on a Large Origin3000 Cluster

5

### Hosts Constituting the Teras Cluster

At present Teras consists of two frames with 512 CPUs and 512 Gigabytes of memory each. One of the two hardware frames is partitioned into five independent hosts, each running a single Unix (Irix) image. The other frame is currently not partitioned and used as a 512 CPU + 512 Gigabyte single host. Thus, the Teras cluster currently comprises six hosts.

Within this cluster of six, three distinct host functions, viz. **services host**, **interactive host** and **batch host**, are discernable:

- 1) **P1**, 32 CPUs + 32 Gigabytes: This host is functioning as the **services host**. It runs the batch server, and provides file server and data migration functions (DMF, TMF). Auxiliary server functions, such as DNS and SMTP relay, are typically

located on this host as well. This host also equipped to serve as the **fail over host for interactive user access**. At present however, this secondary role is disabled

- 2) **P2**, 32 CPUs + 32 Gigabytes: This host is functioning as the **interactive access host**. Users run program interactively here. They develop, compile and test their own codes here. Jobs are submitted to the batch system from here as well. By design, this host also figures as the **fail over host for the services host**. At present however, this fail over role is disabled.

The remaining hosts constitute the batch environment. Users cannot login to batch hosts or the services host. To use a batch host, they must login to the interactive host and submit a batch job. An important detail about the batch hosts, pertaining directly to resource management, is that we do not allocate all available CPUs to the batch system. Rather, **on every batch host we 'reserve' at least four CPUs for the operating system and associated services**.

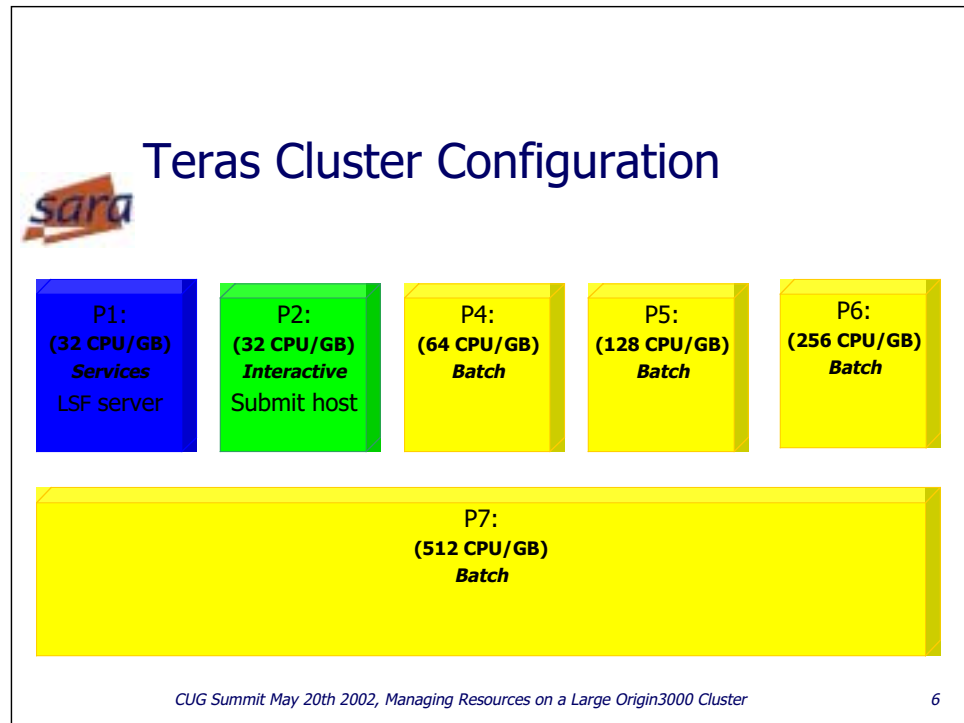
- 3) **P4**, 64 CPUs + 64 Gigabytes: **batch host**. In principle this host is part of the normal production environment. In practice it is sometimes used a 'quarantine' partition, on which pathological jobs can be run either interactively or in batch mode and their behavior can be studied with little or no risk of obstructing the rest of the production.

- 4) **P5**, 128 CPUs + 128 Gigabytes: **batch host**

- 5) **P6**, 256 CPUs + 256 Gigabytes: **batch host**

- 6) **P7**, 512 CPUs + 512 Gigabytes: **batch host**

'Slide 6', below, taken from the presentation, shows the Teras hosts and their role assignments:



## Key Software Components

The key software components on Teras are:

- **Basic Irix Operating System.** We currently use release 6,5,14f. The f-suffix denotes that we use the so-called feature release. We need the feature release for cluster services and clustered XFS file system services (CXFS), but also for other optional Irix components listed below, that play a more prominent role in resource management.
- The batch job scheduling software: **LSF**, version 4.1. This version supports dynamic creation and removal of CPU sets for each batch job by LSF.
- **Irix job limits.** Strictly speaking this is a part of the Irix operating system, but it is an optional component that is only available in the feature release (since 6.510f). Irix job limits support the concept of a 'job container' that contains multiple processes. The job limits apply to the container as a whole.
- **Irix CPU sets.** A workload management tool by which the number of processors that a process or set of processes may use can be restricted.
- **MIPSPRO** software development tools. At present we use release 7.3.1.2m.
- **OpenMP**, a parallelization library and API of compiler directives, using memory objects that are shared between parallel processes, to be used in conjunction with the MIPSpro compilers.
- **MPI**, a message passing parallelization library, an implementation for Irix, available from SGI as a component in the Message Passing Toolkit (MPT). The SGI implementation is optimized for Irix and SGI Origin systems, and is required to confine MPI jobs to an Irix CPU set. We currently use version 1.4.0.2 of MPT.
- **PVM**, a message parallelization library, an implementation for Irix, by SGI, and also part of the MPT. This implementation of PVM is required to run all processes of a PVM jobs within a CPU set.

## Teras Usage

Although users use commercial applications from third party vendors, like Diana or Marc, the vast majority of CPU cycles is spent on codes that users developed themselves. As the system was financed by NCF, it will be no surprise that most users are typically involved in areas of scientific or technological research that have computationally intensive aspects. At present there are about 250 active users from different areas of science.

Users run single CPU jobs as well as parallel jobs. Typical parallel jobs use 8 to 32 CPUs. The majority of the parallel jobs are MPI jobs. A substantial number of jobs uses OpenMP. PVM is used for parallelization as well, but not very often. When we look at the number of jobs, most jobs are single CPU jobs, but when we focus on load, most CPUs are used by parallel jobs.

In proportion to batch usage, interactive usage is negligible. In fact we do not even bother to set up accounting for interactive jobs. The number and size of interactive jobs is of course limited 'naturally', by the fact that users have interactive access only to a single host in the cluster, 'p2', which has 32 CPUs + 32 GB memory.

At present we do not run batch jobs that involve multiple hosts. The biggest batch job that can be submitted in normal production mode is a job using 256 CPUs. Note that only one such a job at a time can run on the Teras cluster, on host 'p7', because each host reserves four CPUs for other purposes than the batch system. Occasionally bigger jobs have been run on Teras, but not in normal production mode, so allocation of resources for such jobs is an ad hoc matter that is not within the scope of this paper<sup>2</sup>.

We can conclude that there is a great variation in types of jobs and also in the resource requirements of these jobs. Some jobs are CPU intensive, other jobs are memory intensive. For a few jobs specific I/O resources have proven to be the limiting factor.

---

<sup>2</sup> To accommodate such rare huge jobs we in fact temporarily exclude the largest host, 'p7', from the batch environment and start the job manually. When the job is done, the host is returned to the batch environment. Thus any job larger than 256 CPUs temporarily monopolizes all resources of host 'p7'. Since draining is not an option with the current version of the batch software (LSF 4.1), the only practical time to do this is right after a maintenance stop of 'p7', when the host is empty anyhow.

## Controlling Resources

### Resource Types and Policy Goals

Despite the noted variation in jobs, any job needs resources from these three categories in varying amounts:

- **CPU resources**
- **Memory resources**
- **I/O resources, including disk space**

We also know from user feedback that **users greatly appreciate reproducible turnaround times of their jobs**. This is especially important to those users that want to do reliable tests on the scalability of their code. The goal of reproducible turnaround times cannot be attained if we one-sidedly focus on optimizing the usage of one resource, e.g. CPU capacity, while neglecting other resources. E.g. the turnaround times of a memory intensive job will vary wildly if we only guarantee the availability of the CPU capacity needed by such a job and cannot control the memory that is actually available when the job runs.

The system administrator wants to control **all** resource types and be able to relate all resource usage by jobs to these jobs. Ideally, the user specifies<sup>3</sup> at job submission time a runtime and all resources that are required by a job during its runtime and sufficient to complete successfully. Ideally, for any job that is admitted into the queuing system, the system administrator subsequently **guarantees the availability of all these resources for the duration of the job**. At the same time the system administrator's task is **to keep the efficient utilization of all resources at a maximum**. Especially in the face of a queue of submitted jobs waiting for execution, resources being idle or used inefficiently should be avoided. Furthermore, the system administrator guards against unplanned monopolization of the system by jobs.

### CPU and Memory Resources

We have already seen that every Teras node is a distributed shared memory system, and that the system has a ratio of 1 GB of physical memory per CPU. Jobs perform best when each CPU uses the local memory that is present in the same C-brick. Therefore we discourage memory over commitment, by tuning job requests to this ratio. E.g. when 8 CPUs are requested for a job, the job will also have 8 GB at its disposal, and is accounted for the usage of 8 CPUs for the duration of the job. Conversely, when a job requests 16 GB of memory, and only a single CPU, 16 CPUs will be reserved nonetheless, and the job is accounted for 16 CPUs for the duration of the job. So, when job submissions explicitly request amounts of memory and CPUs that do not match this 1/1 ratio, the number of CPUs or GB of memory is forcibly<sup>4</sup> adapted to whichever is the highest of the two.

Batch jobs on Teras are accounted on the basis of the wall-clock time that the job takes and the number of processors that are used - in any case reserved - for the job.

---

<sup>3</sup> Of course the specification of minimum requirements in practice may be implicit to the extent that default values have been agreed upon.

<sup>4</sup> Systematic adaptation to the 1 CPU / 1 GB ratio is provided by a wrapper that we have put around the standard LSF submission command.

We do not account memory usage separately and explicitly. But In fact we implicitly account jobs on the number of '1 CPU + 1 GB memory units' for the duration of the job.

### **I/O Resources**

Ideally, I/O resources would be guaranteed like any other resource, up to the amount of disk space that has to be available. In practice I/O resources cannot be specified in a job request. Reading and writing, and all that is needed to do it efficiently, such as managing I/O buffers, is a matter that is delegated to the operating system kernel. The system's I/O load can be related to specific jobs only indirectly.

## **Available Tools for Controlling Resources**

The main tools for controlling CPU and memory resources are the LSF batch system software, Irix CPU sets and Irix job limits. Controlling I/O resources is largely a matter of Irix kernel tuning.

### **LSF**

For the scheduling of jobs, and for limiting the time that a job can run, we use LSF. We do not admit jobs that request a wall clock time over two weeks. On Teras we use LSF version 4.1, which is able to cooperate with Irix CPU sets and Irix job limits.

### **CPU sets**

Irix CPU sets enable us to demarcate a virtual system within a host. CPUs, and, depending on the specification of CPU set configuration parameters, also the memory present on the node boards of the CPUs involved, are grouped for running a particular group of processes – typically a job. Any child process forked off of a parent is automatically attached to the parent's CPU set. I.e.: The scheduling of such children is automatically confined to the CPUs in the parent's CPU set. Thus once the initial process(es) of a job are attached to a CPU set, the entire process tree of a job can be confined to set.

CPU set parameters – or 'tokens' as they are called in the CPU set jargon - that specify to what extent other processes are allowed to use resources within the set are available. Parameters by means of which the opposite can be configured, viz. whether processes associated with the CPU set are allowed to use additional resources outside the set, exists as well. If CPU sets are used, all CPUs that are not allocated to a specific CPU set reside in the so-called 'global CPU set'. CPU 0 of a host cannot be allocated to an exclusive CPU set and always remains in the global CPU set.

LSF dynamically creates one CPU set per job and removes the CPU set when the job is done, returning its CPUs to the global CPU set.

### **Job limits**

Traditionally Unix systems have supported the concept of so-called user limits. User limits however limit the resources associated with a single process. In a batch job environment, where the bulk of resources is used by parallelized jobs that consists of multiple processes cooperating in some way, this is not a suitable tool. Job limits enable us to extend the notion of a user limits to a group of processes and thus apply it to a multi-process job.

Processes are grouped in a so-called 'job container'. All processes that comprise a job are 'in the job container' and are thus collectively confined to whatever limits apply to

the container as a whole. This is a huge improvement over process-based limits. In the context of a job of cooperating processes it is not relevant which process exceeds a specific limit. Moreover the division of labor between cooperating processes does not have to be symmetrical and it is hardly doable to predict which process will need which amount of resources. If one of the process in the container exceeds a limit the container may be destructed, and thus all process within it may be killed. We have noted however that not all exceeding of resource limits that apply to a container lead to a destruction of the container. LSF creates a job container per job, initializes a number of job limits, and then leaves it to Irix to decide what happens if a limit is exceeded.<sup>5</sup>

## Controlling CPU Resources

In the LSF configuration we equate one LSF 'jobslot' with one CPU. When we specify the number of jobslots available on a host, we subtract four from the actual number of CPUs. In this way there always remain at least four CPUs available in the global CPU set for system processes. When a host is 'empty' in the sense that no LSF scheduled jobs are running, all CPUs are in the global CPU set and thus available to all processes. When LSF starts scheduling jobs on the host, the LSF `sbatchd` process takes the required number of CPUs from the global CPU set and creates a CPU set of the requested size for each job.

Our LSF configuration creates **exclusive** CPU sets for its jobs. I.e.: Only processes attached to the CPU set are allowed to run on the set's CPUs and the attached processes will be scheduled on set's CPUs only. Other processes either run on CPUs that belong to other CPU sets that may be exclusive as well, or run on CPUs left in the global CPU set, which is non-exclusive by definition. This is site-specific default behavior<sup>6</sup>.

LSF succeeds quite well at grouping CPUs in reasonable CPU sets because it knows about the NUMALink architecture and Origin 3800 topology. It also runs a 'topology daemon' on every host that monitors the number and location of free CPUs. LSF uses a best-fit algorithm and succeeds quite well at limiting the number router hops between the CPUs in a single set. This minimizes latency in intra CPU set communication and thus contributes to an efficient usage of resources.

A job container per job is created by the LSF `res` process, which can be considered the parent process of a job because it is 'execed' by the `sbatch` daemon child that was forked off for the job.

LSF has defined its own domain in the so-called 'user limits database' (ULDB). The ULDB is a simple ASCII file in which user and job limits can be defined. Multiple limit domains can be defined, and 'LSF' simply is an additional domain that is added to such already existing domains as 'interactive' and '(generic) batch'.

When LSF creates a job container it sets both the current and maximum CPU time limit to the maximum available CPU time defined for a queue. I.e.: These parameters are set at the product of `PROCLIMIT` x `RUNLIMIT`, or number of CPUs x maximum wall-clock time of a job. Usually the set CPU time limit exceeds the wall-clock time requested for a job. For this reason LSF monitors the consumed CPU time and the wall-clock time of a job. When a job exceeds the requested wall-clock time LSF will stop the job.

---

<sup>5</sup> The exception is run time. LSF kills jobs that exceed their run time limits

<sup>6</sup> The default behavior could be changed by specifying different 'tokens' in the `LSF_DEFAULT_EXTSCHED` variable in the `$LSF_ENVDIR/lsf.conf` file

## **Some Problems with Controlling CPU Resources for MPI and PVM Jobs**

With older versions of MPI and PVM that we used in the beginning, there have been some difficulties with controlling CPU resources. This was a result of the way in which these parallel programming environments start up their child processes.

### **MPI**

In these older versions of the MPT toolkit implementation the master MPI processes of a job start all child processes via the array daemon. Thus the array daemon becomes the parent of all MPI child processes of all jobs on the host. These child processes will therefore all be scheduled on CPUs that belong to the CPU set to which the array daemon is attached. The array daemon however is a system process started at system boot time and therefore most probably runs in the global CPU set. This is certainly not what we want.

To solve this, we needed to upgrade to MPT release 1.3 or later. In these later versions CPU set support for MPI was built in. The support for CPU sets consists in the arrangement that not the array daemon but an MPI master process per job is the parent of all of the jobs MPI child processes. In this way all MPI child processes are attached to the right and job-specific CPU set, viz. that of the MPI master process.

### **PVM**

A somewhat similar unwanted situation may be created with PVM jobs. Normally there is one PVM daemon per user per host that is the parent of all other PVM processes of that user on that host. This master process is typically created when the first job of a user on a particular host is started. If the same user starts other jobs on the same host, all processes end up in the CPU set of the PVM daemon, i.e. the CPU set for the first PVM job. With the current PVM implementation this problem can now easily be avoided by setting the environment variable `PVM_MVID`. If this variable is set PVM will start a daemon per job. Thus all child processes will be attached to the job's CPU set.

## **Controlling Memory Resources**

### **Irix Memory Management**

We can look at memory from a hardware point of view and distinguish the following types of memory:

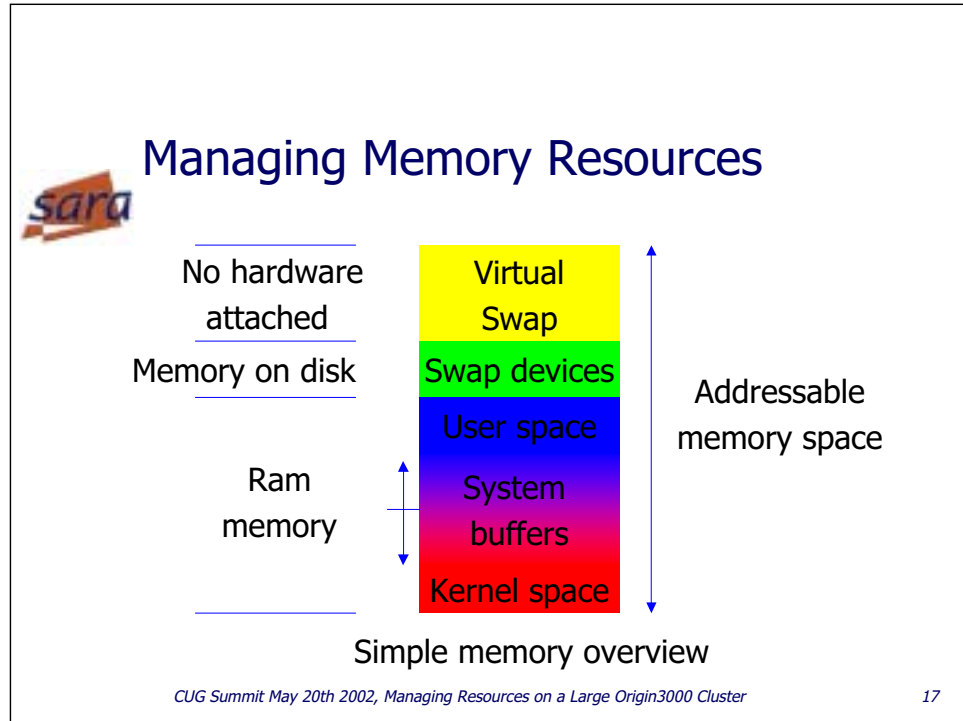
- 1) Physical memory modules located on system boards in C-bricks.
- 2) So-called 'swap space', residing on disk devices, used for paging memory pages temporarily to disk, to create free physical memory.
- 3) So-called 'virtual swap space', which only extends the system's logical address space but is not associated with any hardware device at all. Nothing can be stored there. Virtual swap is used to extend the system's logical address space. This presumes that the address space is used sparsely.

From a software point of view we distinguish the following categories of memory usage:

- 1) Memory statically used by the kernel.
- 2) Dynamically allocated (and de-allocated) memory used by the kernel, mainly used for I/O buffering.

- 3) Memory privately used by user processes.
- 4) Memory shared by two or more user processes.

'Slide 17', below, tries to visualize this overview in a simple map:



On Irix, the dynamic memory allocation functions, e.g. `malloc(3)`, do **not** allocate pages of physical memory, or reserve a part of the swap space. A call to `malloc(3)` only results in the reservation of logical address space. Some counters in the kernel are decremented. The reserved amount of memory is subtracted from the total address space. Physical memory is only allocated on 'first touch'. A logical memory address is only mapped to a physical memory segment when the logical address is used for the first time to actually store data.

An important aspect of management of memory resources is to avoid paging as much as possible, since extensive paging will result in a severe performance hit. Paging is minimized when the memory size of all jobs does not exceed the amount of physical memory.

The memory management characteristics of Irix make it possible – and for some programs this may be very convenient – to allocate a huge amount of memory that is subsequently used very sparsely. Only a few addresses are actually touched and therefore mapped to physical memory. Programs that use this style of memory allocation<sup>7</sup>, may have a large nominal, 'logical', memory size, whereas their actual use of physical memory remains very modest. By consequence, such programs will substantially decrease the system's addressable memory space. Tools to monitor the memory usage of a job report this nominal size of a process as its size. Note that if a number of concurrently running programs actually do this, it is very well possible to

<sup>7</sup> This style of programming is promoted in the Irix programming tutorials and documentation. See e.g. Chapter 1, 'Process Address Space', of the SGI publication 'Topics in Irix Programming' (SGI document number 007-2478-008).

deplete the systems logical address space completely, while there is still a huge amount of physical memory available!

To avoid this problem you can define 'virtual swap' on Irix systems. This is a pure extension of the system's address space without backing it up with any type of actual storage. This way of extending logical memory is limited to one Terabyte. This may seem a lot, but for a host with 512 Gigabytes of physical memory, the limit is actually quite low.

### **General Problems with Shared Memory and the Enforcement of Job (memory) Limits Set by LSF**

Irix has difficulties calculating memory usage on a per job basis. And if it is hard to adequately monitor the different aspects of memory usage, it is even harder to control it in a sensible way. The hardest part appears to be the calculation of shared memory. The only way to do this, is by scanning all memory pages and determine for each page whether it is shared or not, and if so by which processes. This is a very long and busy procedure.

At present all memory usage calculations by monitoring tools are done on a per process basis. The amount of memory used by a single process is recorded without distinguishing between private and shared memory. When the amount of memory used by a group of processes is calculated, the memory usage of the processes involved is just summed. In this way shared memory is not counted once but counted as many times as there are processes sharing it.

LSF runs the `pim` daemon (process information manager) to determine the resource usage of jobs. But, since LSF in principle only uses native system facilities, and Irix has serious limitations in this respect, LSF also has a hard time determining memory usage. In the current version there is however a procedure built into the `pim`, for calculated shared memory. Unfortunately, on a host with 512 Gigabytes of physical memory, this has proven to be a prohibitively long procedure. The `pim` is a single threaded program. Long procedures of this kind lead to blocking of communications with other LSF daemons and therefore to problems with starting new batch jobs.

When a memory limit is specified for a job, LSF applies it as the current resident set size limit (RSS limit) for the job container of that job, i.e.: as the maximum amount of physical memory that all processes of the job taken together are allowed to use. When LSF has initialized the limits it considers the rest up to Irix. It is the operating system's responsibility to detect and respond to any actual occurrences of a job container exceeding the limit. This is not such an unreasonable view, but for the system administrator who that wants to manage memory resources effectively it is problematic nonetheless. At this point the integration of Irix and LSF is far less than desirable, because **Irix simply does not destroy job containers when they exceed their current RSS limit**. As far as Irix is concerned, the memory pages above this limit have only become primary candidates for paging.

In fact, with other limits applied to a job container we have seen that, when a job **is** destroyed by the operating system as a result of a violation of limits, LSF appears to be quite clueless as to why the job (was) finished.

At this moment LSF is not configured to kill jobs when exceeding a memory limit. Via a wrapper around the submit command all memory options are filtered out.

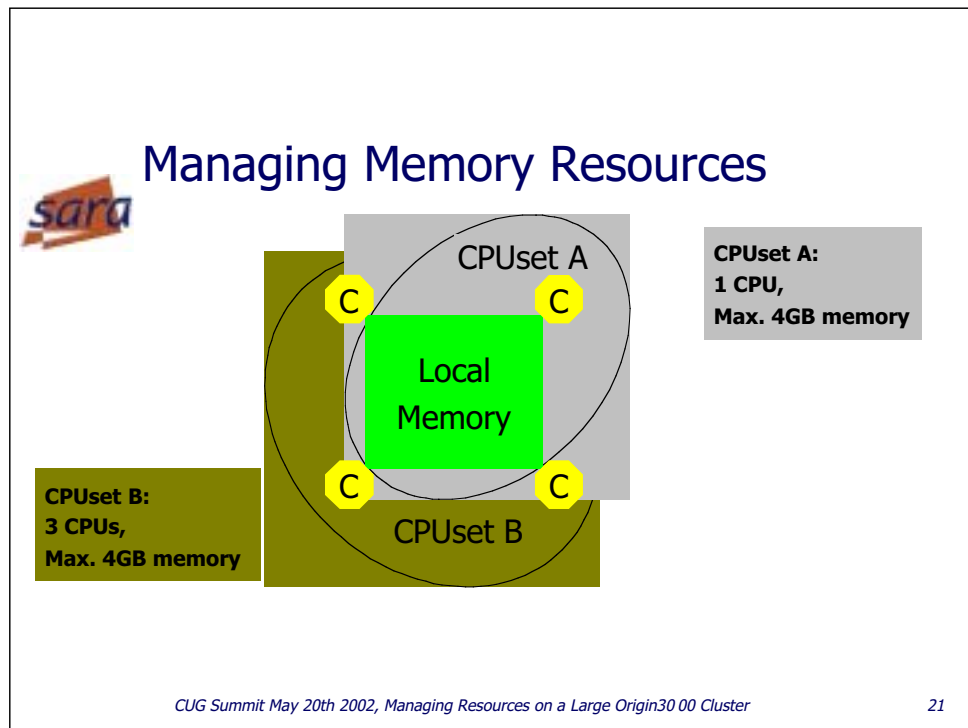
## CPU Sets and Control of Memory Resources

In principle it is possible to regulate memory usage with CPU set parameters. By defining the combination of the CPU set parameters `MEMORY_MANDATORY` and `POLICY_KILL` all processes attached to a CPU set will be killed when one of them tries to access 'far memory'. Far memory is defined as physical memory that is located on C-bricks on which none of the CPUs in the CPU set are located. A C-brick in our configuration we have four CPUs and four Gigabytes of memory. This implies each of the CPUs in a particular C-brick considers these four Gigabytes of memory to be local. So in the context of CPU sets we cannot really treat this as one Gigabyte per CPU. It only makes sense to talk about four Gigabytes per four CPUs.

Because we want to allow single CPU jobs - and any size of job up to 256 CPUs - we will create small CPU sets and CPU sets of sizes that are not multiples of four. Regrettably, this will frequently lead to situations in which we must refrain from using the above mentioned memory control possibilities of CPU sets.

'Slide 21' below, is used to explain this with the help of an example. This picture portrays a C-brick that is a part of a larger host. The CPUs on this C-brick are allocated to two different CPU sets:

- CPU set A consists of 1 CPU
- CPU set B consists of 3 CPUs

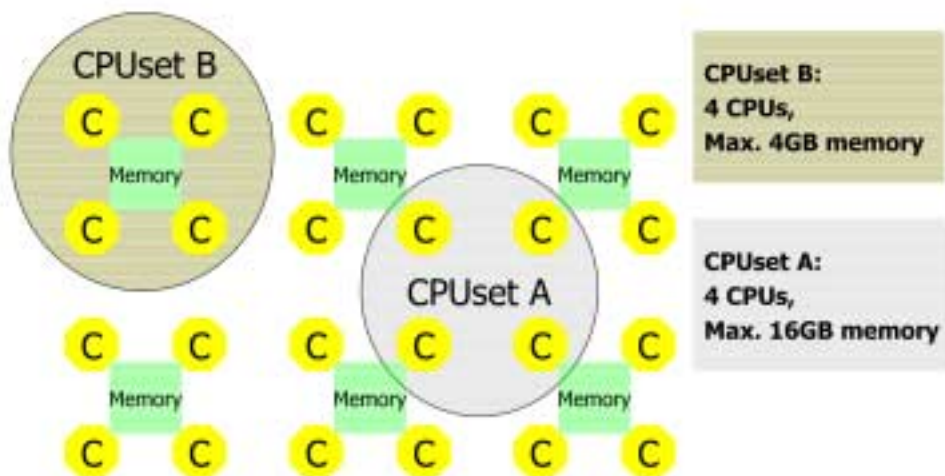


Note that both CPU sets, A and B, will regard all memory located on the C-brick as local memory. Because of this, the single CPU job in CPU set A can allocated up to a maximum four Gigabytes before it has to access far memory - and will be killed. Now suppose the job in CPU set A is already running and has allocated about 3 Gigabytes of memory. This is not what we want, but the real problems start when the job

scheduled on CPU set B starts. We intended to have three Gigabytes of local memory for this job, but there is only one Gigabyte left. When the processes in CPU set B allocate memory over 1 gigabyte, Irix will attempt to allocate 'far memory' because there simply is no local memory left. As a consequence the processes in CPU set B will be killed. The real culprit, the job in CPU set A, gets away with its unabashed behavior of using more memory than requested at submission time and will probably happily run to completion.

This is intolerable and reason enough not to use this mechanism. However, related and similarly awkward situations can arise with larger CPU sets. An example of this is shown in 'slide 22', below.

## Managing Memory resources



*CLUG Summit May 20th 2002, Managing Resources on a Large Origin3000 Cluster*

23

This slide shows a configuration with two CPU sets. Both CPU sets consist of four CPUs. But since one CPU set has all CPUs in the same C-brick while the other set has four CPUs in four adjacent, but different C-bricks, they have very different views on local memory.

CPU set A is distributed over four C-bricks. Because a CPU views the total amount of memory on a C-brick as local memory, CPU set A sees 16 Gigabytes of local memory. If the CPU set is created with the MEMORY\_MANDATORY and the POLICY\_KILL tokens this job will be killed when accessing far memory. This could be only after acquiring 16 Gigabytes of memory. This is much more than the one Gigabyte / one CPU ratio.

CPU set B has all CPUs on the same C-brick. The job will be killed when allocating more than four Gigabytes. This conforms to what we intended.

The allocation of CPUs to CPU sets is done by LSF on basis of a best-fit algorithm. Because we allow single CPU jobs and parallel jobs, and jobs with different run times, together on the same host, fragmentation, non-adjacency, of a host's free CPUs is unavoidable in the long run. We therefore think that the current options of enforcing the usage of local memory by CPU sets are too inflexible. On Teras, we cannot and do not use them at present.

## **Job Limits and Control of Memory Resources**

It is possible to set job limits on memory resources. Because of the way in which SGI has optimized the newer MPI implementation for Origin architectures we cannot use these because virtual memory is not manageable.

The optimization within MPI is that it reserves a shared memory block for process-to-process communications by using the `mmap(2)` memory mapping function. This memory block is about one Gigabyte per (one way) channel in size. The total size needed grows at slightly more than quadratic pace with the number of processes.

We have observed that MPI jobs of four processes need a memory map of about 20 Gigabytes. Larger MPI jobs of eight processes use a map of about 75 Gigabytes. An MPI job of 32 processes needs a map of one Terabyte.

The memory mapping function introduces yet another sparse map between memory as seen by the application and physical memory. The memory map is not the same as virtual memory. Because memory allocation routines, that decrement the amount of unused logical address space, are only called when an offset in the map is actually accessed. Nevertheless the tools for monitoring memory usage (e.g. `top(1)`, `ps(1)`) report it as virtual memory or 'total size of memory'.

We allow testing of small jobs in interactive mode. We have found that when users want to test MPI programs they almost immediately hit the virtual memory limit. To circumvent this we only have a job limit on physical memory.

## **Controlling I/O Resources**

I/O resources have a hardware component and a software component. The hardware component – e.g. controllers, disks, disk striping, etc. – is beyond the scope of this paper. Furthermore we have no reason at all to suspect that I/O performance and resource control problems that we have encountered on Teras are somehow rooted in hardware deficiencies, or a shortage of hardware capacity. The software part of I/O processing is largely a matter of buffer management by the Irix kernel. Thus it is problematic on Irix, like on any other Unix system, to relate I/O load to processes causing it.

Irix uses a dynamic algorithm for allocating and releasing memory for the use of system buffers that are used for caching file system data. These data are of course cached in memory to improve I/O performance. The number of physical reads and writes can be reduced and such requests can be ordered to further optimize physical I/O activities. We have found that for the really large hosts in the Teras cluster the maximum number of system buffer entries (the NBUF kernel parameter) is not big enough for a job set with a number of I/O intensive jobs in the mix. At present we set the NBUF kernel parameter at 1.2 million. We have verified that the hardware is not at the root of the problem. SGI engineers have corroborated our findings. The administration of buffered I/O requests in the Irix kernel cannot deal with these situations.

In the context of resource management the biggest problem is that a few of such jobs ruin the system for all other jobs because they practically monopolize the available buffer entries. The number of buffer entries can only be increased up to a certain point. And in fact, there are also good arguments against increasing it much further: a huge buffer cache is simply substantial usage of memory, on behalf of a few specific I/O intensive jobs, that must nonetheless remain unaccounted as 'kernel overhead'. At present there is no way to limit the load that a single process or job can put on the I/O subsystem.

## Conclusions

On Irix platforms memory and I/O resources are not sufficiently manageable in a large and mixed environment like Teras. On large hosts of 256 CPUs or more we have run into some scalability problems of the I/O subsystem in the Irix kernel.

CPU sets and job limits in principle are excellent tools for resource management. They should be integrated better however, with batch system software like LSF on the one side, and with programming tools like MPI on the other side.

We know that SGI in the process of developing a solution for calculating and monitoring shared memory usage. This will constitute a significant step forward towards better management of memory resources. We think that the ability to define more flexible and to the point physical memory limits for CPU sets is highly desirable as well.

The current state of integration of Irix CPU sets and job limits with LSF is already functional. Particularly the management of CPU resources is functioning quite well and is helping quite a bit reduce the variation in turnaround times of identical jobs. If most jobs in the job mix on a particular host are CPU intensive jobs and they have been submitted with the correct specification of requirements, we are able to attain the goal of reproducible turnaround times.