

# Gang Scheduling for the IBM SP-2 Workstation Cluster

Hubertus Franke, Pratap Pattnaik and Larry Rudolph  
IBM T. J. Watson Research Center  
Yorktown Heights, NY 10598  
email: frankeh@watson.ibm.com

## Extended Abstract

Distributed clusters of workstations interconnected via standard or proprietary packet switching networks have emerged as cost effective alternatives to special purpose parallel computers. These clusters keep up with the progress made in the workstation market by reusing commodity hardware (e.g. processors) and software (e.g. compilers, operating systems etc.). Effective resource management and scheduling is essential for achieving high utilization in such clusters. Nevertheless, there is no one single scheduling strategy that satisfied all the requirements (e.g. priority scheduling, time-sharing, throughput, response time, etc.) of different sites. *Gang scheduling* is a mixed scheduling scheme for distributed systems that combines time-sharing with space-sharing and provides sufficient flexibility to implement higher level scheduling policies. Its benefits are well described in [1]. In gang scheduling, all the activities (processes, threads) of a job ( $\equiv$  gang) are co-scheduled. When a time slice allocated to a job is up, a context switch occurs, resources are reallocated, and other jobs are scheduled.

The IBM SP2 [2] is a workstation cluster that executes IBM's version of standard UNIX (AIX) on each processor and provides interprocessor communication via MPI or IP using the SP2 high performance switch. Achieving communication efficiency in today's parallel systems implies the avoidance of kernel calls by all means. Kernel calls in many of today's operating systems amount between 5 – 20  $\mu$ secs. In IBM SP2, as in many MPP systems, for efficiency reasons, a part of the communication subsystem (CSS) hardware is mapped to the part of the processes address space that is directly addressable in unprivileged state and communication protocols (e.g. MPI, PVM) execute under the control of the application [3]. The resources allocated as part of this mapping are referred to as a CSS window. Thus the context of the processes of a job extends beyond the CPU into the CSS and the adapter. Unfortunately this prohibits several applications to share the adapter simultaneously, which implies dedicating a subcluster to a particular job until its completion.

The objective of our scheduling system for the SP2 cluster is to (a) overcome the single process per processor restriction without compromising performance, reliability or security as accustomed to in the dedicated mode, and (b) to provide a general framework to implement flexible scalable scheduling policies based on

the principle of gang scheduling. The structure of our scheduler framework consists of three collaborating modules (see figure 1): (1) a global scheduler, (2) a local scheduler and (3) an interface to efficiently interact with the CSS.

The **global scheduler** is a pluggable module reacting to normed allocation and deallocation requests from different client interfaces (e.g. interactive, batch). This pluggable scheduler module alone implements the global scheduling policy. Once a job is allocated/deallocated to a set of processors, control messages are sent to the local schedulers to modify their scheduling tables (an example of which is shown in table 1). In order to achieve high scalability and avoid a global context switching signal we require that the local schedulers execute autonomously their schedule. This necessitates that the local clocks are quasi synchronized, which can be achieved with standard Internet NTP timing services. It also mandates that consistency of schedules is enforced by the global scheduler. Messages among the scheduler components are only required when jobs are scheduled or descheduled. To achieve scalability in the control messaging we organize the local schedulers into a hierarchical network (DCMN  $\equiv$  distributed control and messaging network) and require them to perform control message handling and forwarding. If, like in [1], the hierarchy reflects a taxonomy of the scheduling policy, the local schedulers are allowed to spoof on messages (via callbacks) to maintain state related to the hierarchy. The DCMN organization allows communication between all nodes and the global scheduler to be performed in ( $O(\log(\#procs))$ ).

The **local scheduler** on each processor is unaware of any global scheduling policies, it only executes its schedule as generated/manipulated by the global scheduler message. All modules operate based on a global scheduling cycle, a time quantum set by the system administrator and the fraction of each time slice is determined as part of the overall scheduling policy. To avoid the impact of unnecessary paging, we expect this quantum to be a few minutes.

At its heart, the local scheduler requires a **local context switching** service to preempt one process and activate another. To overcome the single application restriction we modified the SP2 CSS kernel extension to allow several applications to attach to the same CSS window in high performance mode. However, since only one application can be active at a

time, the CSS, as part of a context switch, reassigns the access rights to the next process. On SP2 a CSS window includes a job dependent routing table and security key (to be installed in the adapter) and two segments: (1) a virtual packet fifo segment where packets are written and read, and (2) a IO segment to the adapter, to control the adapter-DMA engine to move packets out of the virtual fifo, which for this reason has to be pinned/wired. Because the client's communication library assumes atomicity in various load/store sequences of the low level communication routines, simply stopping (SIGSTOP), and resuming (SIGCONT) a process is not sufficient. The CSS must signal the client (i.e. the communication library) to vacate the CSS and once the client communication library has reached a "secure" state in its communication protocol it yields to the CSS request. This "civilized" manner of vacating the CSS is important since one can not stop the application in the mids of the protocol stack and let it continue later on. A context switch introduces an overhead of 45msec at application level. This mechanism can also be applied in checkpointing to save state. The lower part of figure 1 shows the structure of the extended SP2 CSS and its interaction with the local scheduler and the applications.

To summarize, we allow several applications unrestricted, secured, no overhead access to the CSS, amortizing the cost of context switching the processor and communication state through large grain timeslicing rather than the fine grain UNIX CPU timeslicing.

We have implemented several scheduling policies on top of our scheduler framework. These include deadline scheduling and the distributed hierarchical control strategy described in [1] extended by guarantee of service time. We organize the DCMN into a binary tree and utilize the message spoofing to maintain scheduling hierarchy state at its appropriate level in the tree. This allows the local schedulers to autonomously decide whether to push or hold the cycle of their local scheduling table (see 1). An indepth discussion of the system and performance results are available in a companion paper [4].

## References

- [1] D. Feitelson and L. Rudolph, Gang scheduling performance benefits for fine-grain synchronization. *J. Parallel & Distributed Comput.*, 16(4), pp 306-318, 12/1992.
- [2] T. Agerwala et al., "SP2 System Architecture", IBM Systems Journal, Vol. 34, No. 2, 1995.
- [3] Franke, H., Wu, E., Pattnaik, P., Snir, M.: "MPI Programming Environment for IBM SP1/SP-2" 15th Intl. Conf. on Distributed Computing Systems, Vancouver, 6/1995.
- [4] Franke, H. Pattnaik, P., Rudolph, L.: *Gang Scheduling for Highly Efficient Distributed Multiprocessor Systems*, 6th Symp. on the Frontiers of Massively Parallel Computation, Annapolis, 10/1996.

Class	Fraction $f_i$	Pids	Empty Cycles
3	.50	$pid_1$	-
2	.25	< empty >	DOWN
1	.05	$pid_2$	HOLD
0	.05	< empty >	-

Table 1: *Examples of Local Scheduling Table.*

Each class is allocated a minimum fraction of execution time of each scheduling round. This fraction is static and may be modified by superusers. The Pids field specifies the local processes or gang members of parallel jobs. The Empty Cycles field indicates how the time of an empty class is to be reallocated, i.e. pushed down to lower levels or held at an upper level.

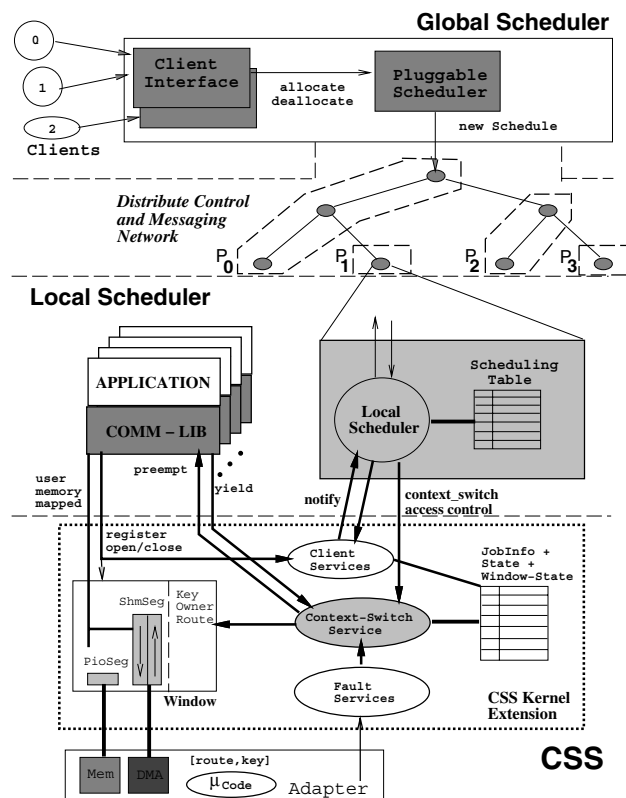


Figure 1: *Overall Gang-Scheduler Architecture.*

A modular hierarchical design customized scheduler components and client interfaces to be integrated into the global scheduler architecture. New schedules are generated by the scheduler and propagated through the a distributed hierarchical communication and messaging network (DCMN) to the participating autonomous local schedulers. The SP2 CSS is extended by context switching services, which preempt/resume the clients and save/restore client state information related to the CSS window, thus enabling time-sharing of a processor while maintaining efficiency and protection of dedicated system.