Plagued by Work: Using Immunity to Manage the Largest Computational Collectives

Lucas A. Wilson[1], Michael C. Scherger[2] & John A. Lockman III[1]
[1]Texas Advanced Computing Center, The University of Texas at Austin
[2]Texas A&M University – Corpus Christi
United States

1. Introduction

Modern computational collectives, ranging from loosely-coupled Grids and Clouds to tightly-coupled clusters, are progressively increasing in both capability and complexity. This has created a need for more efficient methods to schedule tasks to hosts. Typically, system resources in these environments are managed with a combination of simple heuristics and bin-packing algorithms to perform common operations such as backfill. However, as the size and scope of these computational collectives grows ever larger, different approaches must be employed to cope with both the number of resources to manage and the volume of jobs to schedule. One possible avenue is to distribute the task of managing these massive-scale systems across the participants, giving each resource a say in how the final scheduling solution will appear.
The introduction of multi-/many-core architectures has complicated the problem of performing effective scheduling on large-scale systems. The number of allocatable elements per system is now increasing at a staggering rate as hardware manufacturers attempt to keep pace with Moore's Law (Moore 1965). In clusters, for example, the number of "nodes" - standalone physical systems with a network connection - has stabilized due to limitations in current switching technology and power/cooling capacity. However, each node now has more allocatable cores, increasing the cores per node "density" of the system overall. Scheduling algorithms will be required to cope with scheduling quantities of elements increasing by orders of magnitude every few years, while still providing timely decision information.
The Asynchronous Lymphocytic Agent-based Resource Manager (ALARM) was first proposed as a novel method of distributing the task of managing a large set of resources by mimicking the actions of the mammalian immune system (Wilson 2008). Previously reported results demonstrated the viability of using the immune system as a metaphor for distributed resource management and provided a comparison of ALARM to other, more widely-recognized scheduling heuristics (Scherger 2009).
In this chapter we detail how the scheduling problem can be described in terms of the mammalian immune system and provide a description of the ALARM method. We provide comparative results against common scheduling heuristics on large-scale

simulations of a tightly-coupled parallel cluster, as well as an analysis of the networking overhead created by using ALARM on the same simulations.

2. Background

As in many tightly or loosely coupled distributed systems, process scheduling is an integral component in determining the efficiency of a high performance computer system. Continuing research in process scheduling algorithms is conducted to ensure that sub-systems in high performance computing will be able to simultaneously maximize utilization and ensure process completion in a specified time period.

Scheduling plays an important role in distributed systems in which it enhances overall system performance metrics such as process completion time and processor utilization (Tel 1998). There are two main classes of distributed process scheduling algorithms: sender-initiated and receiver-initiated algorithms (Chow 1997). A third class of distributed process scheduling algorithms is the hybrid sender-receiver algorithm and is a compromise to overcome the problem from the two algorithms (Ramamritham 2002).

The role of a distributed process scheduler is the same as normal scheduling: improve system performance metrics (Audsley 1994). In distributed systems the existence of multiple processing nodes is a challenging problem for scheduling processes onto processors. One cause for this complex problem is that process scheduling must be performed locally and globally across the whole system. A process created at a node can move to other nodes in the system to redistribute work load as to achieve an improved system performance. Global scheduling performs load sharing between processors. Load sharing allows busy processors to load some of their work to less busy, or even idle, processors (Boger 2001).

Load balancing is a special case of load sharing. In load balancing the global scheduling algorithm is to keep the load even (or balanced) across all processors (Malik 2003). Sender-initiated load sharing occurs when busy processors try to find idle processors to load some work. Receiver-initiated load sharing occurs when idle processors seek busy processors (Stankovic 1999). While load sharing is worthwhile, load balancing is generally not worth the extra effort. Small gains in execution time of tasks are offset by extra effort expended in maintaining a balanced load.

In a distributed system individual nodes have their own policy for determining when to accept or remove tasks. The characteristics of the distributed scheduling algorithm are normally depended on the reason of its existence such as information exchange, resource sharing, and increased reliability through replication and increased performance through parallelization (Boger 2001). Scheduling algorithms have four distinct policies: the transfer policy, the selection policy, the location policy, and the information policy. The transfer policy decides when a node should migrate a particular task, and the selection policy decides which task to migrate. The location policy determines a partner node for the task migration, and the information policy triggers and contains the collection of system state from all nodes: when, what and where (Chaptin 2003).

Scheduling algorithms can also be classified as static or dynamic (Tel 1998). These decisions are based on task characteristics and the current system state. Scheduling algorithms that use a static approach calculates (pre-determine) schedules for the system. It requires a-priori knowledge of the tasks characteristics and does not require any overhead at run-time. Scheduling algorithms that use a dynamic approach determines schedules at run-time which provide a flexible system that can adapt with non-predicted events. Dynamic scheduling algorithms have a much higher run-time cost overhead but can give greater processor utilization.

Comparison of scheduling algorithms has been researched by (Tel 1998) to evaluate the performance between sender-initiated policy and receiver-initiated policies. Their results prove that sender-initiated policy is better than receiver-initiated policy in light to moderate system loads while receiver-initiated policy is better than sender-initiated policy in high system loads. In addition, (Ramamritham 2002) and (Audsley 1994) have conducted a study towards the performance of sender-initiated and receiver-initiated policies in both homogenous and heterogeneous distributed system with regards to First Come First Serve (FCFS) and Round Robin (RR) scheduling policies. Apart from that, the study also includes the impact of variance in job service times and inter-arrival times. (Boger 2001) provides the explanation on performance sensitivity of the sender-initiated and receiver-initiated policies, to three factors: node-scheduling policy, variance in job inter-arrival, while (Chaptin 2003) has reported the performance of several load sharing policies based on their implementation of both sender-initiated and receiver initiated policies on a five node system connected by a 10Mbps communication network.

Alternatively, (Stankovic 1999) has conducted a study and compared the sender-initiated, receiver-initiated and hybrid (it is called symmetrical-initiated in that literature) policies pertaining to system workload and the effect of probing to overall system performance.

3. Scheduling Tasks on Large-scale Distributed Systems

In general, the scheduling problem is NP-Complete, meaning that a guaranteed optimal solution cannot be found in polynomial time (Cormen 2001). As a result, many resource managers schedule tasks by either building a scheduling matrix (processors x time-window) (Fig. 1) and using an algorithm to solve this packing problem in order to most-efficiently (although not optimally) allocate tasks within that particular time window, by using less expensive heuristics, or through a combination of both. These approaches typically require categorizing tasks into classes of importance.

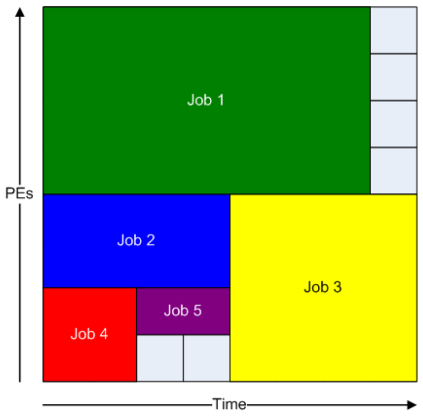


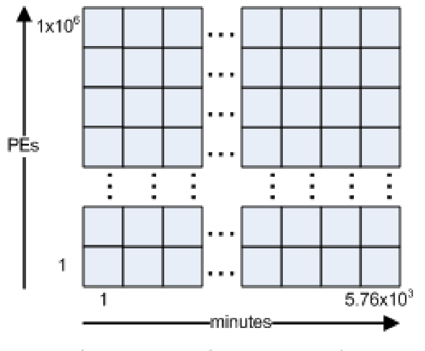
Fig. 1. Example Scheduling Matrix

This is done by either having multiple bins for tasks (e.g. multiple queues on a batch system) or by using a series of priority rules and weighted functions to generate a multi-objective importance factor that can be used to reorder tasks.

3.1 Matrix-based Scheduling Approaches

Managers that use a scheduling matrix have several obvious weaknesses, although they are most likely to generate near-optimal solutions. The primary problem is that they are static in nature. Like many centralized algorithms, solutions to the packing problem can only be performed given the information present when the algorithm begins execution. If new tasks arrive while the algorithm is running, those tasks must either be ignored until the next scheduling period or the algorithm must be restarted. This process can also become extremely expensive both computationally and spatially. Most scheduling algorithms tend to be written with dynamic programming or greedy approaches, the computational costs of which are *O(n)* (Sadfi 2002) and *O(n²)* (Hwang 1991) on small computational sets, respectively. It is important to note, however, that these solutions are pseudo-polynomial in nature, meaning that although they provide solutions in a polynomial fashion for small cases, at extremely large scale they are still NP-Complete (Garey 1979).

Generating complete matrices requires $p*t$ memory locations, where p is the number of processing elements (PEs) in the system and t is the size of the scheduling window. This presents an enormous scaling problem, as the only options when increasing the size of the system (p) is to either increase the amount of memory consumed or reduce the size of the scheduling window (t). As many systems have execution policies that allow for maximum runtimes of 48 hours or more, this typically requires reducing the resolution of the time axis (i.e. changing the smallest time element from 1 minute to 15 minutes). Reducing the resolution will degrade solution quality by creating pockets of idle time on the system, and increasing available memory is a costly alternative, thus limiting the effectiveness of this particular scheduling approach on massive-scale machines exceeding 100,000 or even 1,000,000 PEs.



$1x10^6 * 5.76x10^3 = 5.76x10^9$ cells in matrix

Each cell contains a single 8 byte long integer

$5.76x10^9 * 8 = 46.08x10^9$ bytes of memory

Fig. 2. Memory Requirements for Large-scale Scheduling Matrix

As Fig. 2 demonstrates, scheduling a 1,000,000 PE system using a 96 hr window (which would allow 2 back-to-back 48 hour jobs to be scheduled) with 1 minute resolution would require $1x10^6 * 96 * 60 = 5.76x10^9$ matrix locations. If each matrix location needed to store an 8-byte long integer, such as a job ID, then the scheduling matrix would need to be $5.76x10^9 * 8 = 46.08x10^9$ bytes, or 46.08GB.

3.2 Heuristic-based Scheduling Approaches

Unlike matrix-based scheduling algorithms, heuristic scheduling approaches tend to require less computational and spatial overhead. However, like all other centralized algorithms, they are inherently susceptible to failures in system components that may drastically alter how jobs need to be coordinated. Additionally, many heuristics tend to be static in nature, unable to account for jobs that arrive after the scheduling algorithm has begun. These techniques are typically used in conjunction with matrix-based approaches in batch-processing systems, where weighted functions are used to generate multi-constraint priorities to determine job execution order.

4. Artificial Immune Systems (AIS)

Research into the usability and effectiveness of AIS has been ongoing for the last decade. Although AIS is a relatively new concept in the field of nature-inspired computing, it already shows remarkable ability to adapt to extremely dynamic environments and is well suited to distributed applications (de Castro 2002). Many existing systems are based on the clonal selection model, and very closely resemble other evolutionary computation techniques, most specifically genetic algorithms and genetic programming (Cutello 2002, de Castro 2000).

Because of the noticeable parallels between protecting the body and protecting networks, AIS have been widely used in the field of network security and access management (Boukerche 2004, Kim 2001). Relying heavily on Immune Network Theory (INT), these AIS solutions analyze typical network traffic patterns, determine when abnormal traffic is on the system, then alerts managers to possible security risks. Although some work has been done in automatically protecting network systems from intrusion, many AIS solutions are simply for the detection of abnormal traffic, and not for blocking out the intruder.

4.1 Immune Network Theory

Scientists first believed that the mammalian immune system developed immunities to infection through one process – clonal selection. Clonal selection resembles the evolutionary process, with many thousands of white blood cells created through the act of cloning an existing, activated white blood cell. During the cloning process, called clonal expansion, these cells undergo "hypermutation," making the antibodies on the surface of these cloned cells different from the source. The "affinity" of these clones – the ability of these cells to identify the set of antigens of the infection currently being combated – is then established, and those with the greatest affinity survive while the others are destroyed. By repeating this process again and again when an infection was present, immunity to that infection would eventually be found (Jerne 1955).

The first major theoretical shift in the operation of the immune system came in the 1970's with Immune Network Theory (Jerne 1974). Unlike clonal selection, which creates antibodies through a method of repeated affinity determination and hypermutation, INT attacks new antigens by building up complex antibodies from smaller, more basic antibodies. Like clonal selection, INT relies on linking random combinations of antibody building blocks together to form a single immunity. However, the building blocks in INT are much larger than in clonal selection, reducing the time required to find an appropriate immunity.

4.2 Danger Theory

Danger Theory is a debated concept in immunological research that looks at how the immune system can identify potential problems not by attacking things that are foreign, but by attacking only those things which create "danger." According to danger theory, chemical signals released when a cell is damaged are received by nearby antigen-presenting cells, and then carried to local lymph nodes (Matzinger 1994). The strength of these chemical signals weaken with distance, and because a certain threshold is required for white blood cells to recognize these signals, a set region, or "danger zone," exists around the site of the incident. When antibodies in the lymph nodes "match" antigens collected from within the danger zone, the corresponding B-cells are activated and undergo clonal expansion in order to combat the infection (Aickelin 2002).

5. Applying the Immune System Metaphor to the Scheduling Problem

With all nature-inspired meta-heuristics, a mapping of naturally occurring phenomena to concepts and events in the problem space first must be performed to successfully apply the lessons and processes of the natural system to the target problem. The mammalian immune system consists of myriad chemicals, cells and organs working in concert with one another to perform the task of destroying or preventing infections. (a) Several infections likely occur simultaneously, (b) the immune system must cope with the fact that infections can be spread out over the entirety of the mammalian body, (c) infections cannot all be treated by the same immunological response, and (d) new infections may appear at any time.

The scheduling of tasks in distributed memory environments presents the same type of situational difficulties as dealing with infections in the body. (a) There may be many tasks to be scheduled simultaneously, (b) tasks may be parallel in nature and need resources from many of the distributed memory resources in the system, (c) many tasks have hardware or software dependencies that require the scheduler to act accordingly by ensuring that those tasks are mapped to locations that can accommodate the dependency requirements, and (d) the task space is extremely dynamic, with many new tasks being generated at any given time.

5.1 Defining a Set of Terms for an Immune System-based Resource Manager

Tasks can clearly be seen as infections from the perspective of a distributed-memory system. The job of the resource manager is then to complete as many tasks, or kill as many infections, as possible. This means that the system or environment to be managed can be likened to the body; each task that is submitted to the system must be executed, just as each infection that enters the body must be destroyed.

To accomplish the goal of destroying infections entering the body, the immune system makes use of special blood cells known as lymphocytes. Although the biological model contains many types of lymphocytes that perform various sets of actions, for the purposes of applying this metaphor to resource management they can all be considered a single type of entity. In a distributed-memory environment, the individual resources that compose the system are responsible for executing tasks.

All infections have a set of chemical "hooks" on their surface called antigens. Conversely, each lymphocyte contains a chemical marker known as an antibody. The job of the immune system is to create an immunological response that properly maps a series of lymphocyte antibodies to the sequence of antigens on the infection. A resource manger, whether controlling a homogeneous or a heterogeneous environment, must similarly map the resource requirements of a task to the appropriate set of resources to effectively execute that task.

Based on these astonishing similarities between the mammalian immune system and the operating requirements of resource managers, we can create a set of terms that frame the distributed-memory environment, its individual resources, and the tasks it executes in the context of the immune system. Table 1 defines the terminology set that will be used.

Immunological Term	**Resource Management Term**
Body	System to be managed
Lymphocyte	Resource in the system
Infection	Task to be executed
Antigen	Resource requirement of task
Antibody	Resource capability

Table 1. Defined terms of immune system metaphor

5.2 Defining Events and Responses for an Immune System-based Resource Manager

Now that a set of terms has been established that places the scheduling problem in the context of the immune system, we must define both the events that occur over the life-cycle of a resource manager, as well as the appropriate responses by the resource manager to those events in the context of the immune system metaphor. Although there are many differing and competing theories on how the immune system both detects malicious activity and responds to that activity, we will use a combination of two theories that fit best with our distributed management environment. Danger Theory provides a simple, distributed method for performing the detection component, while INT gives us a simple method for forming proper responses to those detected events.

Every scheduler must contend with a series of different time-independent events: (a) A task being submitted to the system, (b) a task beginning execution on the system, (c) a task completing execution on the system, either successfully or in error, and (d) resources becoming available or unavailable. Each of these events can be difficult for static, centralized scheduling algorithms to contend with, as they would require the re-execution of the algorithm using a new snapshot of the system.

One of Danger Theory's central concepts is the use of chemical messages to detect the presence of malicious entities. In a biological system, the distance from which an event can be detected is limited due to the decay of these chemical signatures as they travel through the bloodstream. This message-based approach employed in the Danger Theory model can be applied to a system of distributed resources connected via the network. Although a computer network is not limited in the distance it can send messages (through the use of intermediate relays), it would not be beneficial to saturate the network with broadcast messages every time an event occurs.

If one were to use network messages to signal the occurrence of events, it would allow a system to provide dynamic, real-time reactions to those events. Each independent agent in the system (infections and lymphocytes) would be responsible for both transmitting and reacting to various message signals propagated via the network. If each message was transmitted to only a limited subset of the entire network, it would allow many independent events and reactions to occur simultaneously without adversely affecting one another.

In the mammalian immune system, lymphocytes are alerted to the presence of an infection when a victim cell is destroyed and releases a particular chemical signature. Consequently, the infectious agents of an immune system-based resource manger (tasks) would be responsible for the transmission of a message to signal their own presence. Since chemical signatures decay over time and distance, only a limited number of lymphocytes would be close enough to receive that signature and respond to it, or within the "danger zone," as it is referred to in Danger Theory. As a result, only a limited number of lymphocytic agents (system resources) nearby the signaling infectious agent should be privy to this message.

This limited message distance has several interesting side-effects that can be advantageous to a resource manger. If "distance" is measured by some network metric (e.g. hops), then nearby resources will most likely be better localized (such as on the same switch in a switching hierarchy), and therefore provide better communications performance for tightly-coupled parallel codes. Additionally, since only a small number of resources are immediately alerted to the presence of an infection, the likelihood of saturating the network with response messages is reduced. Lastly, because large parallel tasks will be unable to secure enough resources to begin execution immediately, the immune system-based approach provides a natural form of "backfill," which maximizes utilization by squeezing smaller jobs into the slots leftover from scheduling larger jobs. Because large jobs cannot immediately consume available resources, smaller tasks can begin execution while the large jobs are acquiring the resources necessary to execute.

Once a lymphocyte has been alerted to the presence of an infection in the mammalian immune system, it must mount some form of immunological response. In INT, this response would consist of T-cells carrying infection associated antigens back to lymph nodes, which would then begin generating antibodies which match all or part of the antigen pattern. This partial pattern-match allows the immune system to begin the process of mounting a response to infection before a complete, perfect solution is discovered. The generation of partial solutions and iterative construction of solutions is crucial in distributed systems as the individual components do not have the ability to constantly or consistently communicate with one another. In an immune system-based resource manager, lymphocytes which receive a signal from an infection would check to see if any of their antibodies, or resources, match any of the antigens, or resource requirements, presented by the infection. If so, the lymphocyte would respond by binding itself to the infection.

Although immediate response works well when a lymphocyte is idle and unbound, what happens when a lymphocyte is busy or bound to another infection? In the case of a lymphocyte being busy, it should ignore the message. In most cases preemption is not desired on large-scale systems, so there should be no reason to stop executing a task to handle another one. In the case of a lymphocyte being bound to an infection but not running a task, one of two actions could be taken: (1) The lymphocyte could decide that the infection it is currently bound to has higher precedence, and ignore the incoming request, or (2) the lymphocyte could decide that the new infection has higher precedence, and switch from being bound to the first to being bound to the second. By choosing from these actions, a simple priority system can be developed within the resource manager with little computational overhead on the part of the lymphocytes, which are also responsible for executing tasks.

After an infectious agent has received a response from a lymphocyte, it will associate that binding response with a particular antigen subset, indicating that those pieces of the solution have been discovered. When the entire antigen set has been associated with a binding lymphocyte, the infectious agent will signal the lymphocytes associated with that solution to begin execution. When this occurs, the lymphocytes will begin execution of the binary or script associated with that infectious agent.

Unfortunately in many cases an infectious agent cannot receive enough binding responses after the first signaling, either because there are insufficient resources within the danger zone, or those resources are busy executing other tasks. In a biological system, the effect of an insufficient immunological response would be the spreading of the infection to other cells or parts of the body. This has the effect of increasing the size of the danger zone, as more chemical signals are created as the infection spreads. In an immune system-based resource manager, the spreading of an infection can be accomplished not through the replication of the infectious agent, but by increasing the size of the danger zone surrounding the infectious agent. Instead of being able to signal only the most local lymphocytes, an infection would then be able to signal lymphocytes beyond those, up to a certain limit. Theoretically, this limit could expand to the size of the system, if no sufficient response is provided in a timely fashion.

Once a task begins execution, it will continue to execute until "completion," defined as successful or in error, or until an external signal requires that it terminate, such as through user request or the extinguishing of a preset time limit. The completion of a job, regardless of return code, can be considered normal termination. Conversely, the termination of a job through user request, extinguishing of a preset time limit, or by other external means can be considered abnormal termination. Cells in a biological system also terminate in normal and abnormal fashions. Normal cell death is defined as necrosis, whereas abnormal cell death is defined as apoptosis. We shall use the same nomenclature to describe the completion of tasks in the immune system-based resource manager.

When a task completes, the lymphocytes executing that task will transmit a message back to the infectious agent denoting that the task terminated normally, via necrosis. When a task is terminated by external means, the infectious agent will notify the lymphocytes executing that task that the task terminated via apoptosis. The lymphocytes will then

respond back to the infectious agent in the same manner that they would for normal termination. In both cases, the lymphocytes will transmit back the return code of the task along with the appropriate signal. When an infectious agent receives termination signals from all associated lymphocytes, the agent will complete and the task will be considered done.
Now that we have a complete picture of the life cycle of an infectious agent, from the moment it appears on the system to the time it terminates, we can see a relatively small set of signals are exchanged between infections and lymphocytes in order to successfully execute tasks. Table 2 and Table 3 define the signals that will be needed for an immune system-based resource manager.

Signal Name	Definition
SIG_INFECT	Indicate the presence of an infection
SIG_ATTACK	Notify lymphocytes that the associated task should be executed
SIG_APOPTOSIS	Notify lymphocytes that a task should be terminated immediately (abnormal termination)

Table 2. List of infection-produced signals

Signal Name	Definition
SIG_BIND	Notify an infectious agent of intent to execute
SIG_DELAY	Notify an infectious agent that it will be binding to another infection
SIG_NECROSIS	Notify an infectious agent that the associated task has completed/terminated

Table 3. List of lymphocyte-produced signals

5.3 Design of Autonomous Agents

With both a working set of terms and a series of events, signals and responses defined, we can begin the process of designing the two types of autonomous agents that form the core of an immune system-based resource manager. Both infections and lymphocytes would be represented as autonomous agents, with each resource having a single lymphocytic agent and each job being "wrapped" in an infectious agent.
Each infectious agent resides on one of the various compute resources in the system, and makes elementary decisions based on response messages received from lymphocytes. Fig. 3 details the design of an infectious agent.
Each resource houses a single lymphocytic agent, which responds to messages from various infectious agents. When a lymphocytic agent receives notification of an infectious agent’s presence (via SIG_INFECT), it must also check its antibody list to ensure that is has at least one of the necessary resources to execute that job. Fig. 4 outlines the design of a lymphocytic agent.

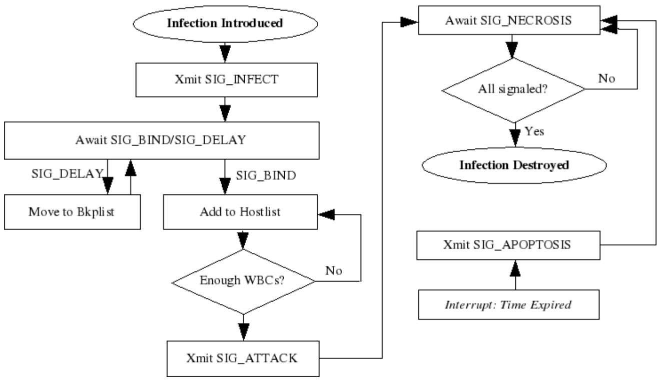


Fig. 3. Control flow graph of infectious agent

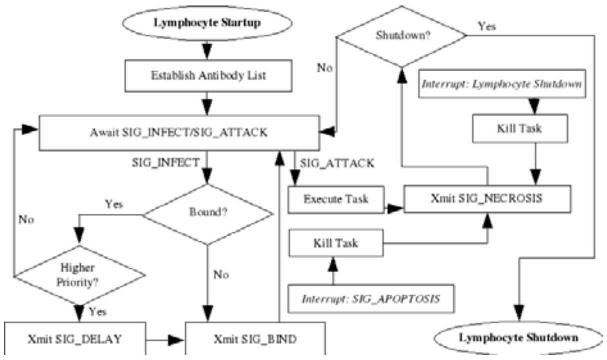


Fig. 4. Control flow graph of lymphocytic agent

5.4 Design of Signal Messages

In order for the various autonomous agents to communicate with each other, they must be able to exchange messages over the network. Each message must be small, so as not to interfere with other user-based network traffic, while containing sufficient information to effectively perform the scheduling operations.

Each message must contain some identifier of the type of signal being transmitted. Additionally, some messages need to send auxiliary information. SIG_INFECT must contain the antigen list in the message, to allow lymphocytes to determine whether or not they should participate in the solution. Also, SIG_NECROSIS must also contain the return code of the task(s) in order to provide that information back to the infectious agent. Each UNIX return code is an integer from 0 to 255, allowing it to be encoded in 8 bits. Additionally, since only 3 bits are required to encode all 6 signal types, the remaining 5 bits of that byte can be used to encode the antigen list, or resource requirements, of an infection. An example message layout is given in Fig. 5.

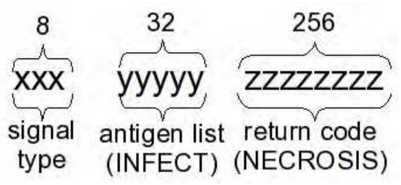


Fig. 5. Example message packet

6. Experimental Validation

To help us evaluate the potential benefits and pitfalls of this immune system-based approach to managing large-scale resource collectives, a series of simulations were performed to help identify performance in two major areas: schedule generation and network congestion.

6.1 Evaluating Schedule Quality

Although it can be difficult to quantify the "quality" of a schedule, there are several metrics that can be used to provide comparisons. By comparing these metrics against schedules generated by other techniques, we can create a picture of the approximate quality of schedules produced. Six metrics (Table 4) were used to compare schedule quality against three basic scheduling heuristics: Smallest Job First (SJF), Largest Job First (LJF), and Best Fit First (BFF).

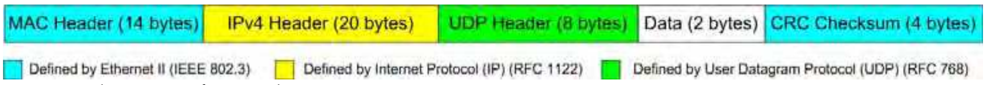
Metric	Definition
Throughput	Avg. number of jobs completed per hour
Turnaround time	Avg. time between job submission and completion
Wait time	Avg. time between job submission and execution
Load Balance	Std. Dev. in number of jobs per node
Utilization	Ratio of in-use cores to total cores
Makespan	Time from submission of first job to completion of last job

Table 4. Scheduling metrics and definitions used in simulation study

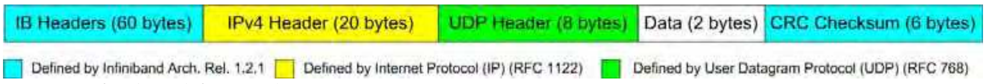
6.2 Evaluating Network Congestion

Although distributing the scheduling problem eases the computational requirements, it can possibly have adverse affects on network performance, either by consuming bandwidth or by overloading the network with excessive small messages. Our tests will examine the aggregate number of signals of each type, in five-minute windows, and then calculate the overall bandwidth and load burdens on two different networking technologies – Gigabit Ethernet and Infiniband (IB).

Ethernet II-based User Datagram Protocol/Internet Protocol ver. 4 (UDP/IPv4) packets consist of a 46 byte message header (IEEE 2005, Braden 1989, Postel 1980) plus a payload section, which for our purposes would house the two byte message illustrated in Fig. 5. This means that each message transmitted using IP over Ethernet would be 48 bytes in length (Fig. 6).


MAC Header (14 bytes)
IPv4 Header (20 bytes)
UDP Header (8 bytes)
Data (2 bytes)
CRC Checksum (4 bytes)
Defined by Ethernet II (IEEE 802.3)
Defined by Internet Protocol (IP) (RFC 1122)
Defined by User Datagram Protocol (UDP) (RFC 768)
Fig. 6. Ethernet II frame description

In order to send the same UDP/IPv4 message over IB, the IP and UDP packets must be embedded into a native IB frame (known as IP over IB). To have a multi-network, globally addressable IB message, 66 bytes of header and CRC information are required (Infiniband 2007). When combined with the previously described 28 bytes of IP and UDP headers plus the 2 byte message illustrated in Fig. 5, the total size for a UDP/IPv4 over IB message comes to 96 bytes (Fig. 7).


IB Headers (60 bytes)
IPv4 Header (20 bytes)
UDP Header (8 bytes)
Data (2 bytes)
CRC Checksum (6 bytes)
Defined by Infiniband Arch. Rel. 1.2.1
Defined by Internet Protocol (IP) (RFC 1122)
Defined by User Datagram Protocol (UDP) (RFC 768)
Fig. 7. Infiniband frame description

7. Results

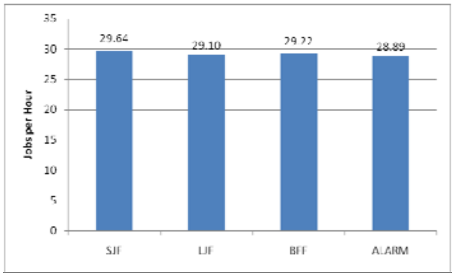
A simulated 4,096-node, single core per node cluster built on a discrete, event-driven engine was tasked with executing 100,000 jobs submitted at a rate of one every sixty seconds. The jobs used in this job deck were taken from the execution logs of the Lonestar Dell-Linux cluster at the Texas Advanced Computing Center (TACC) in Austin, Texas. Each job ranged in size from a single core (serial) job to 1,024 cores and had execution times up to 48 hours.
Each infectious agent simulated had an expansion period of thirty (30) seconds, meaning that every half minute, an infectious agent's danger zone was expanded to include two more resources in a linearly-arranged list of the 4,096 nodes.

7.1 Schedule Quality Comparisons

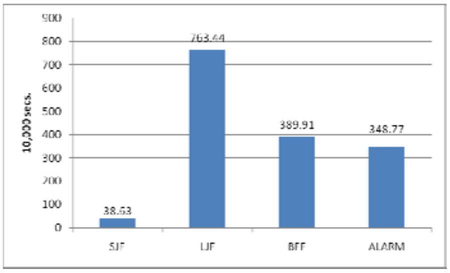
Fig. 8 shows the results of the previously described simulation runs and how the ALARM method compares to the three basic heuristics (Scherger 2009). Although ALARM was not the top performer, it was able to compete with all three comparison heuristics, placing second in both the turnaround time and wait time. The only significant downside for the immune system-based method was in load balance, although this was most likely caused by persistent saturation of the scheduler with new jobs. With the three comparison heuristics, rate of submission does not affect the resulting schedule generation, while changes in submission rate can affect the binding policies of lymphocytic agents to infectious agents.

7.2 Network Congestion

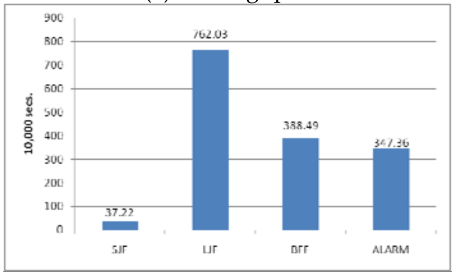
When offloading computation into the form of communication, latency and bandwidth become a topic of great importance which must be investigated. To validate this method we looked closely at the time period where the largest number of messages were generated by ALARM. Fig. 9(a) shows the results of the previously described simulation runs and focuses on the aggregate number of signals generated by the ALARM technique, spanning 144 simulated days. On the 80th day of this simulation ALARM generated a peak number of signals demonstrating a period of full system saturation where the number of signals sent totaled 51,818,685.



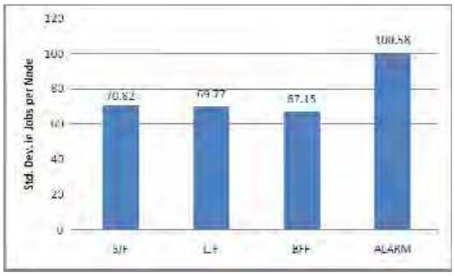
(a) Throughput



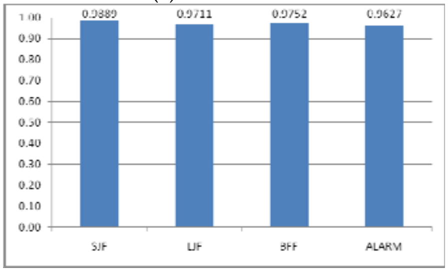
(b) Turnaround Time



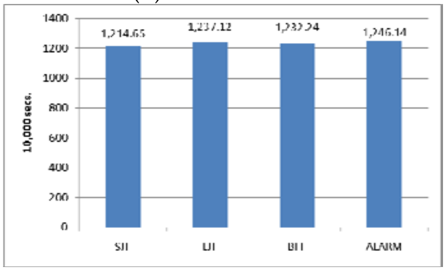
(c) Wait Time



(d) Load Balance



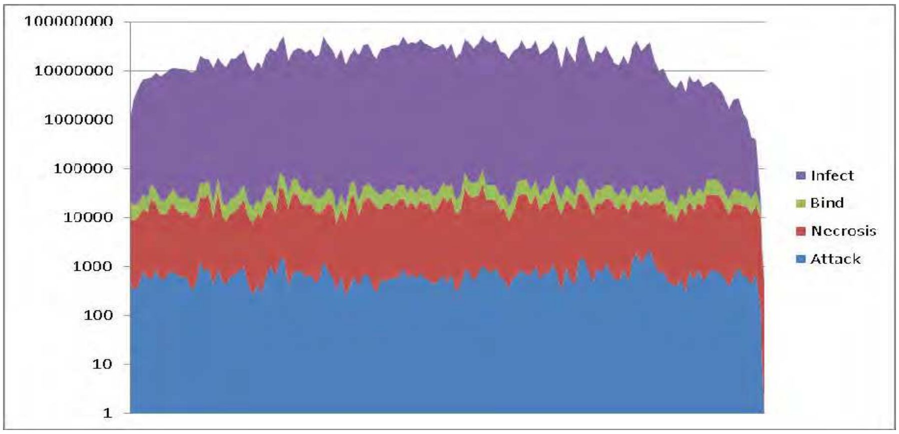
(e) Utilization



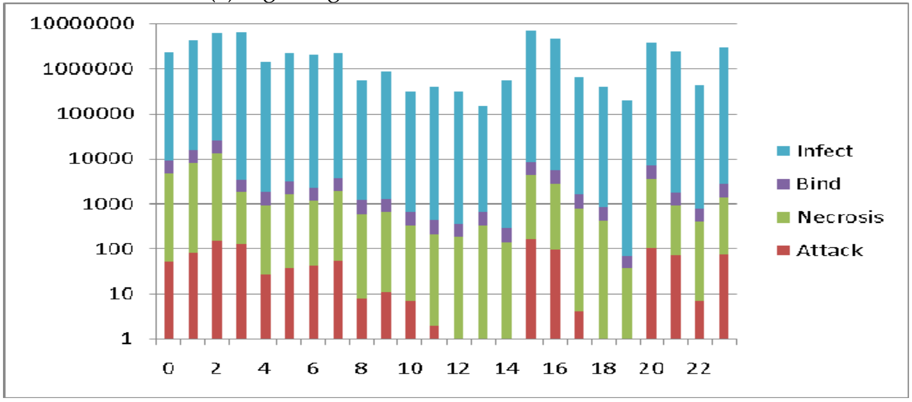
(f) Makespan

Fig. 8. Schedule quality comparisons

Fig. 9(b) provides a closer examination of the 80th day divided into one hour windows, showing that in the 15th hour ALARM generated approximately 6,400 signals per second. Latency of Gigabit Ethernet has been measured between two machines at 135 µsec (Farrell 2000). Using the figures from the peak of our simulation run, the ALARM method would utilize 86.4% of the available network frames, while utilizing 0.2% of theoretical peak bandwidth. InfiniBand, with a latency of 1.5µsec (Koop 2008) , would utilize 0.96% of the available network frames while utilizing 0.05% of theoretical peak bandwidth.



(a) Signals generated over simulation lifetime

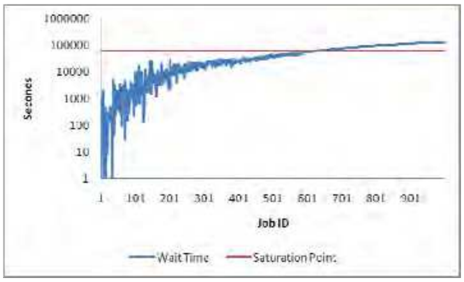


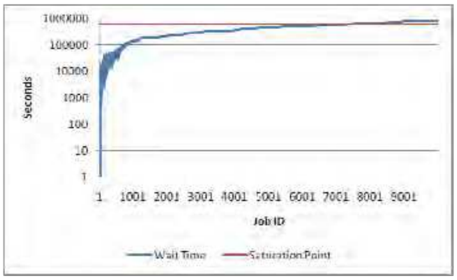
(b) Signals generated on peak day (day 80)

Fig. 9. Network signals produced by ALARM

7.3 Pitfalls of Trivial Decision and Expansion Strategies

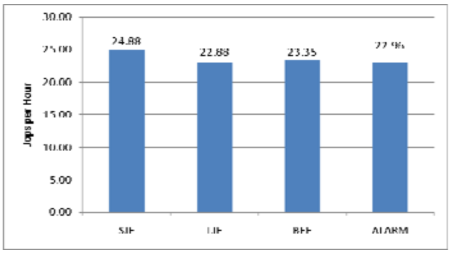
ALARM ranked last in nearly all of the metric categories, due mainly to the limitations inherent in the simple heuristic tiebreaker chosen for lymphocytes. Each lymphocyte – or resource on the system – used a job ID-based priority for determining which of many simultaneous SIG_INFECT messages to respond to. In small cases, this can be a very simple and effective tiebreaker, favoring older jobs over newer jobs. However, as the wait time of all jobs increases, the ALARM scheduling method reaches an absolute saturation point where the wait time of each infection submitted exceeds the amount of time necessary for it's influence to expand to the entire system. For example, the simulation system has 4,096 PEs, and each infection increases its danger zone by a radius of 1 PE every 30 seconds, meaning only 61,440 seconds (17 hrs.) are required for an infection's danger zone to encompass the entire system. When this point is reached, lymphocytes that complete jobs are immediately bombarded with SIG_INFECT requests from all currently active infections, and each lymphocyte therefore chooses the infection with the smallest job ID. This makes the entire system behave like the simple heuristic "First Come-First Served" (FCFS), causing large sections of the system to remain idle periodically as resources are allocated to very large jobs without considering smaller jobs behind them. Reducing the expansion rate would alter this behavior temporarily, although a saturation point would eventually be reached that again causes this job ID-based priority heuristic to resort for FCFS. As Fig. **10**10(a) shows, by job 600 the system had already achieved this level of saturation.

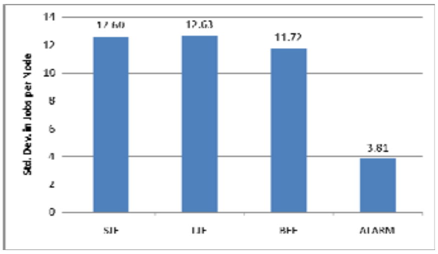




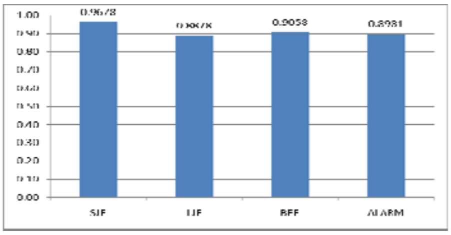
(a) 100,000 jobs with 30 sec. expansion (b) 10,000 jobs with 300 sec. expansion
Fig. 10. Wait time per job approaching saturation point

Additional simulations on the same size system using a reduced expansion rate of once every five minutes (300 seconds) on a smaller instance (10,000 jobs) of the same job deck used for this experiment were done on all four scheduling methods, with ALARM performing significantly better in all 6 categories and generally outperforming all metrics except for SJF (Fig. 11). This reduced expansion rate delayed complete system saturation until jobs maintained a minimum wait time of 307,200 seconds (3.5 days) (Fig. 10(b)). Future investigations into the use of various tiebreaker heuristics and their effects on overall system behavior could be beneficial in improving the performance of ALARM in production settings.

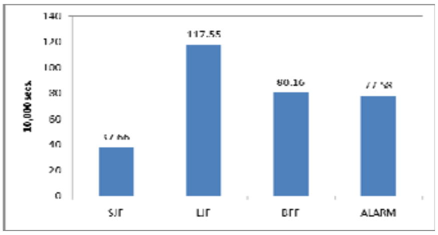




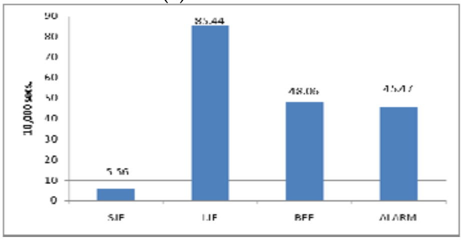
(a) Throughput (b) Load Balance



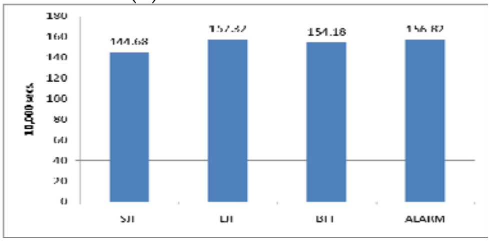
(c) Utilization



(d) Turnaround Time



(e) Wait Time



(f) Makespan

Fig. 11. Schedule quality comparisons for 10,000 job / 300 sec. expansion case

Additionally, more intelligent signaling and expansion systems for infections could also be explored to determine if more complex network-based algorithms (e.g. back-off algorithm in TCP/IP) could be beneficial in improving the overall performance of ALARM in large-scale production environments.

8. Future Work

As we have demonstrated, a distributed scheduling method - based on the functionality of the mammalian immune system - can indeed be a viable, scalable solution for generating timely scheduling information with limited computational and communications overhead. In our current tests, lymphocytic agents used a trivial decision strategy (lowest job-ID first) for making binding decisions. However, additional investigation into improved decision strategies could lead to more efficient scheduling information without creating additional overhead, thus helping to possibly improve load balance or reduce total makespan. Investigation into improved expansion strategies on the part of infectious agents may also aid in reducing communications overhead.

So far, all investigations have been through simulation in order to verify the feasibility of using an immune system-based scheduling method on large-scale systems. However, the design and development of an actual resource management tool based on this approach should be a primary focus of efforts going forward. Once an initial system has been developed, further research into various decision and expansion strategies can be tested on real-world tasks and hardware. Additionally, development of a real-world system will allow research to concentrate on many of the other components of resource management tools besides the scheduling engine, such as statistics gathering and reporting, administrative control of resources, fault recovery, etc.

9. Conclusions

Historically, increases in computational performance have been achieved by chip manufacturers shrinking transistor scale and increasing clock speed. This meant that although overall performance continued to increase, the number of allocatable elements in a system remained relatively constant. Today, with the ever-increasing popularity of computational collectives ranging from Grids and Clouds to clusters and the increase in unit density with the advent of multi-/many-core architectures, computational performance is achieved by increasing the number of allocatable elements instead of increasing the individual performance of each of those elements. For schedulers and resource managers, this poses a fundamental problem - at what point will traditional, centralized techniques become inadequate for scheduling jobs on massive-scale machines encompassing 100,000 or possibly 1,000,000+ PEs?

As we have seen with high-performance computing in the last decade, the solution to improving performance is to distribute the workload across multiple resources. Meta-heuristics, such as artificial immune systems, have been demonstrated as viable solutions to solving complex computational problems in large-scale, dynamic environments. ALARM, the Asynchronous Lymphocytic Agent-based Resource Manager, uses this immune-system metaphor to create a distributed, dynamic solution to scheduling jobs on large scale computational collectives, whether loosely- or tightly-coupled.

Results presented here and in other works (Wilson 2008, Scherger 2009) demonstrate the viability of this approach and suggest that implementation of a real-world system based on this technique would be a reasonable near-term goal. Additional investigation into lymphocyte decision strategies and infection expansion strategies may also yield higher quality results without significant additional computational or communications cost.

10. References

Aickelin, U. and Cayzer, S. (2002). The danger theory and its application to artificial immune systems. *Proceedings of the 1st International Conference on Artificial Immune Systems*, pp. 141-148.

Audsley, N. and A. Burns, 1994, Real -Time Scheduling, in Department of Computer Science, University of York.

Boger, M., 2001, *Java in Distributed Systems*, Wiley.

Boukerche, A, Juca, K., Sobral, J.B. and Notare, M. (2004). An artificial immune based intrusion detection model for computer and telecommunications systems. *Parallel Comput.*, 30(5-6), pp. 629-646.

Braden, R. (ed.) (1989). Requirements for Intenet Hosts -- Communication Layers, *Network Working Group Request for Comments (RFC) 1122*, Internet Engineering Task Force, October 1989.

de Castro, L.N. and von Zuben, F.J. (2000). The clonal selection algorithm with engineering applications. *Artificial Immune Systems*, pp. 36-39.

de Castro, L.N. and Timmis, J. (2002). *Artificial Immune Systems: A New Computational Approach*. Springer-Verlag, London, U.K.

Chaptin, S.J., 2003, Distributed and Multiprocessor Scheduling, University of Minnesota.

Chow, R. and T. Johnson, 1997, *Distributed Operating Systems and Algorithms*, Addison-Wesley.

Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L. and Stein, Clifford (2001). *Introduction to Algorithms*, Second Edition. MIT Press. 2001.

Cutello, V. and Nicosa, G. (2002). An immunilogical approach to combinatorial optimization problems. *Proceedings of the 8th Ibero-American Conference on AI*, pp. 361-370.

Farrell, P.A. and Ong, H. (2000). Communication performance over a gigabit Ethernet network, *Proceedings of the Performance, Computing, and Communications Conference*, pp. 181-189.

Garey, M.R. and Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

Hwang, Cheng-Tsung, Lee, Jiang-Humg and Hsu, Yu-Chin. (1991). A Formal Approach to the Scheduling Problem in High Level Synthesis. *IEEE Transactions on Computer-Aided Design*, 10:4, 1991.

IEEE (2005). IEEE Std 802.3-2005, IEEE, 2005.

Infiniband (2007). *Infiniband Architecture Specification*, 1, rel. 1.2.1, November 2007.

Jerne, N. K. (1955). The natural selection theory of antibody formation. *Proceedings of the National Academy of Science*, USA. 41, 1955, 849-857.

Jerne, N.K. (1974). Towards a network theory of the immune system. *Ann Immunol (Paris)*, 125C(1-2), 1974, 373-389.

Kim, J. and Bentley, P.J. (2001). An evalutation of negative selection in an artificial immune system for network intrusion detection. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 1330-1337.

Koop, M.J., Jones, T., and Panda, D.K. (2008). MVAPICH-Aptus: Scalable high-performance multi-transport MPI over Infiniband, *Proceeding of the International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1-12.

Malik, S., 2003, *Dynamic Load Balancing in a Network Workstations*, Prentice-Hall.

Matzinger, P. (1994). Tolerance, danger and the extended family. *Annu. Rev. Immun.*, 12:991, 1994.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

Postel, J. (ed.) (1980). User Datagram Protocol, *Request for Comments (RFC) 768*, USC/Information Sciences Institute, August 1980.

Ramamritham, K. and Stankovic, J.A. (2002). Dynamic Task Scheduling in Hard Real-Time Distributed Systems, *IEEE Software*, 2002. 1(3): p. 65-75.

Sadfi, C., Penz, B. and Rapine, C. (2002). A dynamic programming algorithm for the single machine total completion time scheduling problem with availability constraints. *Eighth international workshop on project management and scheduling*, 2002.

Scherger, M. and Wilson, L. A. (2009). Task Scheduling Using an Artificial Immune System in a Tightly Coupled Parallel Computing Environment. *The 2009 International Conference on Genetic and Evolutionary Methods (GEM'09)*, July 2009.

Stankovic, J.A. (199). Simulations of three adaptive, decentralized controlled, job scheduling algorithms. *Computer Networks*, 1999. 8(3): p. 199-217.

Tel, G.,(1998), *Introduction to Distributed Process Scheduling*, University of Cambridge.

Wilson, L. A. (2008). Distributed, Heterogeneous Resource Management Using Artificial Immune Systems. *Proceedings of the International Parallel and Distributed Processing Symposium, NIDISC*, Apr. 2008.

Parallel and Distributed Computing
Edited by Alberto Ros

ISBN 978-953-307-057-5
Hard cover, 290 pages
Publisher InTech
Published online 01, January, 2010
Published in print edition January, 2010

The 14 chapters presented in this book cover a wide variety of representative works ranging from hardware design to application development. Particularly, the topics that are addressed are programmable and reconfigurable devices and systems, dependability of GPUs (General Purpose Units), network topologies, cache coherence protocols, resource allocation, scheduling algorithms, peertopeer networks, largescale network simulation, and parallel routines and algorithms. In this way, the articles included in this book constitute an excellent reference for engineers and researchers who have particular interests in each of these topics in parallel and distributed computing.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Lucas A. Wilson, Michael C. Scherger and John A. Lockman III (2010). Plagued by Work: Using Immunity to Manage the Largest Computational Collectives, Parallel and Distributed Computing, Alberto Ros (Ed.), ISBN: 978-953-307-057-5, InTech, Available from: http://www.intechopen.com/books/parallel-and-distributed-computing/plagued-by-work-using-immunity-to-manage-the-largest-computational-collectives