Application of Lagrangian Receding Horizon Techniques to Resource Management in Ad Hoc Grid Environments

Ralph H. Castain[1] William W. Saylor[1] H.J. Siegel[1,2]
ralph.castain@colostate.edu *wsaylor@earthlink.net* *hj@colostate.edu*

Colorado State University
[1]Department of Electrical and Computer Engineering
[2]Department of Computer Science
Fort Collins CO 80523 USA

Abstract

An ad hoc computing grid is characterized not only by constraints on the available energy and communications bandwidth associated with each participating device, but also by the dynamic nature of the grid itself. This is caused by the mobile nature of the assets connected to the grid (computing devices, sensors, and users), plus the fragility of interconnecting communication links. The challenge, therefore, is to efficiently and robustly manage both computational and communication resources in this dynamic, unpredictable environment. This paper reports on one potential solution that combines Lagrangian techniques with the receding horizon concept used in modern robust control systems.

I. Introduction

A heterogeneous network of mobile computing devices, data collection sensors, communication channels, and multiple users can be used as an *ad hoc* computing grid to solve large-scale tasks. Such grids are characterized not only by constraints on the available energy and communications bandwidth associated with each device, but also by the dynamic nature of the grid itself. Assets connected to the grid (computing devices, sensors, and users) can – and frequently do – appear and disappear from the grid at unanticipated times. In addition, communication links are prone to spurious failures and occasional noise that significantly impacts the grid's ability to transfer information between nodes. Further compounding the problem is the lack of full advanced knowledge of the characteristics and arrival time of tasks that will be submitted to the grid.

The challenge confronting *ad hoc* grid systems is to efficiently manage both computational and communication resources in this dynamic, unpredictable environment while ensuring that adequate quality of service (according to some appropriate metric) is maintained. Meeting this challenge will require a resource manager that can rapidly reschedule tasks "on-the-fly" when computing resources are unexpectedly lost or become unavailable for an unpredictable period of time. The ability to recover partial results would be a desirable feature, but may prove too costly in light of its attendant demands on the communication subsystem.

One approach to solving this problem assumes full advance knowledge of all tasks that will be executed by the system during some specified period of operation. Such "static" (or "offline") resource managers schedule all the tasks and communications prior to any actual execution taking place. Thus, static resource managers can take advantage of this knowledge to, for example, schedule data transfers from one task to another in anticipation of the second task becoming ready for execution. In addition, static resource managers have more time to compute task/machine assignments due to their offline operation.

In contrast, "dynamic" resource managers operate "online" and hence do not have access to the full range of information available to their static counterparts. Dynamic resource managers must schedule tasks and communication while execution of previously scheduled tasks is taking place, thus placing added emphasis on the execution speed of the heuristic itself. In addition, dynamic heuristics cannot take advantage of visibility into the future to pre-schedule actions such as communications.

This paper reports on one potential dynamic solution to the *ad hoc* grid challenge that is based on combining the "receding horizon" concept from the control community with Lagrangian-based resource management methods found in some modern manufacturing line scheduling systems. In this initial investigation, the behavior of sev-

eral possible variations of the proposed solution were studied in a simplified situation as a first step towards developing an overall solution to the problem. The results of these tests were subsequently compared against the performance from a common static heuristic.

II. Related Work

The Lagrangian approach to constrained problems was initially developed as a method for dealing with systems involving nonholonomic constraints – i.e., constraints that cannot be expressed as a function of the form $f(r_1, r_2, r_3, \ldots, t)=0$ [Go50]. Examples of nonholonomic constraints include the walls of a container that constrain the motions of a gas, or a particle constrained to move on the surface of a sphere. In these situations, the equations governing the behavior of the system can be solved by introducing the constraints into those equations through the use of Lagrangian multipliers – undetermined parameters that are often functions of time.

Since that time, the Lagrangian technique has been applied to an ever-increasing range of problems that involve complex system equations and nonholonomic constraints [LeM67] [BrH75]. Extending the technique to the resource management problem where heuristics must be bound by "hard" constraints on precedence, energy, and time – all nonholonomic in nature – was a natural fit.

The most common use of the Lagrangian technique for scheduling has been in the manufacturing area. In these cases, the problem is to optimally schedule the manufacture of parts in an industrial operation where the parts require multiple machining operations and have desired completion times. There is a penalty for finishing too early or too late. The machining operations are constrained by the available resources in terms of the types of machines used, the finite number of machines of each type, and the scheduling precedence requirements.

Luh and Hoitomt [LuH93] successfully adopted the Lagrangian approach by breaking the overall manufacturing problem into a series of sub-problems. This approach combines Lagrangian relaxation techniques with post-solution scheduling heuristics. The Lagrangian multipliers attempt to measure the marginal costs associated with machine capacity and scheduling precedence constraints. As dynamic changes in the manufacturing environment occur, the pre-existing optimal values of the Lagrangian multipliers can be used as a starting point for the computations for the changed environment. However, the resulting Lagrangian-based solutions typically represent infeasible schedules in that there are temporal machine capacity and scheduling precedence constraint violations. The final solution requires the use of list scheduling techniques whereby a greedy scheduling heuristic is used to sequentially schedule tasks by order of decreasing marginal costs. The problem is effectively treated as a scheduling solution to a static environment and the Lagrangian multipliers cannot be modified, and generate a guaranteed feasible solution, in a continuously changing environment.

Subsequently, Luh et al. [LuZ00] extended the Lagrangian technique through the introduction of a neural network that adjusted the Lagrangian multipliers as the scheduling progressed to achieve closer-to-optimal results. The key result in this work was a proof of the convergence of the Lagrangian relaxation neural network (LRNN) approach for constrained optimization problems. There are no requirements on the differentiability of the describing functions or the continuity of the decision variables – provided that the evolution of the decision variables leads the Lagrangian to a global minimization.

This approach works well for statically mapping the part manufacturing sequence in an industrial environment. However, it has two significant limitations:

(a) the LRNN formulation generally converges to a solution that may not be entirely feasible because of the difficulty in characterizing some constraints. As a result, there is a final step that readjusts the mapping to ensure the final solution is feasible; and

(b) as implemented, the technique cannot respond to dynamic changes in the production line, such as the loss of a machine, without rescheduling the entire production sequence.

Removing these limitations requires the introduction of a second technique, also taken from the controls community. The receding horizon concept for optimal control first appeared in the 1960s as a method for accurately controlling nonlinear processes (e.g., [LeM67]). As described at that time, the basic concept involved the measurement of the current process state, followed by rapid computation of the predicted evolution of that state for a finite period of time in the future (known as the horizon). The results of the computation were then used to control the process during the horizon period until a new measurement of the process state is made. This procedure is repeated throughout the process, thus providing a piecewise solution to the control problem.

Little was done with the concept initially, primarily due to the high computational requirements involved in the prediction portion of the algorithm. However, the concept resurfaced in the late 1970s in the form of Model Predictive Control (MPC) – also known as Receding Horizon Control (RHC) – as microprocessors began to enter the market. Variants of the technique rapidly spread, particularly in the petrochemical industry, where eco-

nomic considerations push for the absolute maximum in production.

Since that time, MPC has emerged as a standard technique for control of multivariable, constrained systems [WhS92]. However, although there are many similarities between the problems of optimal control of dynamic systems and the scheduling of production tasks in a manufacturing environment where there are numerous precedence and resource availability constraints, the application of MPC techniques to the problem of resource management has only recently become the subject of research.

In particular, Pereira and Sousa [PeS97] developed a hierarchical computational framework for the integrated planning and scheduling of manufacturing operations. The manufacturing operations modeled are discrete events that have been treated as a continuous flow model at multiple levels of a hierarchical structure. The optimization occurs with respect to a distinct number of operations that define the current point of the receding horizon. Combinations of levels of hierarchy, the length of the time horizon, and degree of task aggregation are used to analyze the trade-off between complexity and model accuracy. In the scheduling determination, the operations of the machines are first ordered by criticality. Then a feasible schedule is constructed for the most critical machines and the allocation and scheduling of the next lower level of machines is generated. Mathematically, the optimization step uses a pseudo-Hamiltonian formulation of the continuous flow problem.

This approach appears to work well for well-described manufacturing operations where the parameter perturbations are small. The approach does not demonstrate the ability to be implemented in a real-time computational architecture with large dynamic changes in the rate of task arrivals or machine and communication path disruptions.

This paper explores an alternative approach to the existing literature by explicitly pursuing a computational strategy that lends itself to real-time allocation and scheduling decisions based upon a continuously updated optimization problem. The fundamental problem addressed is one of optimal allocation of resources in a constrained, dynamic, sometimes unreliable, computing grid. The proposed solution combines the receding horizon concept with Lagrangian multipliers to find the optimal solution with respect to a global cost function with explicit constraint formulations. This approach is particularly attractive due to its potential for being mapped directly onto hardware such as digital signal processors (DSPs) or field-programmable gate arrays (FPGAs), thus providing the algorithmic speed required for deployment in many real-time field applications.

III. *Ad Hoc* Grid Environment

The purpose of this initial study was to assess the performance of the mapping heuristics in three different, but static, *ad hoc* grid configurations. Dynamic reconfiguration of the grid between the three cases was not permitted during this initial work. This allowed the effort to focus solely on the behavior of the heuristics in each configuration, without complicating the problem with the transition response.

In each case, the heuristics were tested using an application consisting of a single task composed of $|T|=1024$ communicating subtasks. Since both the energy available and the time for the entire task to complete within the *ad hoc* grid was limited, each subtask was provided with two versions (a "primary" and a "secondary" version) that could be executed. The secondary version of each subtask used 10% of the energy and time of the primary, or "full", version, and transferred 10% of the data output to subsequent child subtasks. This secondary version represents a reduced capability designed to provide some lesser overall value while consuming correspondingly fewer resources than its primary counterpart. Thus, the secondary version allowed the resource manager a greater range of options to successfully map the complete task, especially when faced with very tight energy and time constraints.

The estimated time to compute (ETC) each subtask was calculated using the Gamma distribution method described in [AlS00]. $ETC(i, j)$ represents the estimated execution time of subtask i on machine j. For this study, ten different ETC matrices were developed with a mean estimated execution time for a single subtask of 131 seconds. Each ETC matrix included two classes of machines (fast and slow). Fast machines, on average, executed roughly ten times faster than slow machines. The exact ratio was determined randomly for each subtask to avoid any deterministic influence.

The objective of each experiment was to maximize the number of subtask primary versions (T_{100}) that could be executed while still completing the application within specified time and energy constraints. These were subsequently chosen to force load balancing across all available machines. The subtask dependencies were given by a directed acyclic graph (DAG). Ten separate DAGs were used in the simulations, each generated using the method described in [ShC04].

In addition, the size of the global data item $g(i,j)$ that each subtask communicated to another subtask was specified. The size of each $g(i,j)$ was generated according to the method described in [ShC04]. These values were not varied across the three *ad hoc* grid configurations studied

in this effort. The different combinations of ETC and DAG pairs, therefore, provided 100 unique scenarios that were used for all three cases.

Each of the three grid configurations was distinguished by a different number of machines $|M|$, as shown in Table 1 below. Case A represented the baseline grid configuration where all machines are present. Case B was created by eliminating one of the slow machines, while Case C eliminated one of the fast machines.

Table 1. Simulation configurations

Configuration	# "Fast" Machines	# "Slow" Machines
Case A	2	2
Case B	2	1
Case C	1	2

Each machine *j* was characterized by four parameters:

i. the energy capacity of its battery, $B(j)$;
ii. an energy consumption rate for computation per time unit, $E(j)$;
iii. an energy consumption rate for communication per time unit, $C(j)$; and
iv. a communication bandwidth, $BW(j)$.

The values of these parameters for both fast and slow machines are shown in Table 2. These values represent a rough industry average based on microprocessors and battery capacity on selected, currently commercially-available machines. Fast machines were typified by the Dell Precision M60 notebook computer using an Intel MP4M processor operating at 1.7GHz. The numbers for the slow machines were taken from typical personal digital assistant (PDA) computers such as the Dell Axim X5 that uses an Intel PXA255 processor operating at 400MHz.

Table 2. The values of $B(j)$, $C(j)$, $E(j)$, and $BW(j)$ for fast and slow machines

	"Fast" Machines	"Slow" Machines
$B(j)$	580 energy units	58 energy units
$C(j)$	0.2 energy units/sec	0.002 energy units/sec
$E(j)$	0.1 energy units/sec	0.001 energy units/sec
$BW(j)$	8 megabits/sec	4 megabits/sec

The time required to transmit one bit of a global data item from machine i to machine j was given by:

$$CMT(i,j) = \frac{1}{min(BW(i), BW(j))}$$

Using these energy capacities and consumption rate characteristics for the individual machines, a value of 34,075 seconds was selected as the time constraint (τ) for completing execution of the full task based on experiments using a simple greedy static heuristic. Requiring the task to complete within this time constraint, subject to the available energy and consumption rates shown in Table 2, forced the resource managers to balance the load across all available machines.

Several simplifying assumptions were incorporated into the simulations:

(a) machines consume no energy when idle, nor do they consume energy when receiving information. Machines only consume energy when computing and when transmitting information to another machine. No energy is consumed when transferring information between subtasks on the same machine;
(b) each machine is capable of executing only one subtask at a time. Communication, whether transmitting or receiving, does not interfere in any way with execution;
(c) each machine can simultaneously handle one outgoing data transmission and one incoming data reception; and
(d) subtasks are available for scheduling as soon as their input data is available (i.e., when their parent(s) complete execution). Subtasks cannot begin execution, however, until all of their input data is transmitted from the parent(s) computing machine and received by the machine to which the subtask is assigned.

IV. Simplified Lagrangian Receding Horizon (SLRH) Resource Manager

The Simplified Lagrangian Receding Horizon (SLRH) resource manager owes its name to its dual heritage from the Lagrangian and receding horizon concepts. The term "simplified" signifies that the Lagrangian multipliers are treated as constants as opposed to adjusted during the course of the simulation. This results in a less optimal mapping, but provides a faster and easier implemented algorithm. Assessing the impact of this simplification was a key element of the study.

The SLRH is fundamentally constructed as a dynamic heuristic – i.e., the algorithm only operates on subtasks that can be mapped at any given moment, without knowledge of the full range of subtasks in the simulation. In a truly dynamic environment, each subtask would arrive at some non-deterministic time. For simplicity in this study, each subtask was assumed to be available for mapping as soon as its precedence constraints had been satisfied.

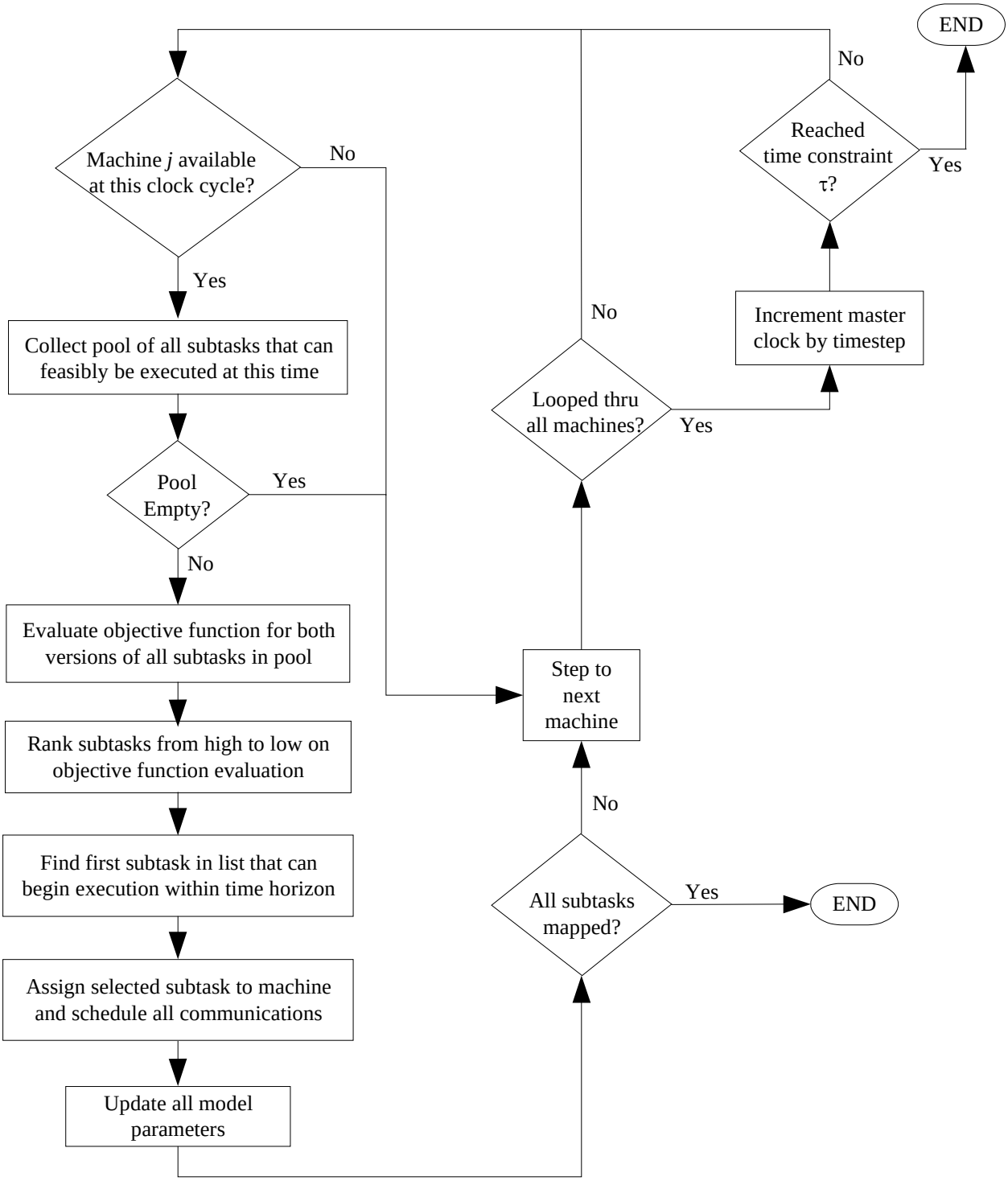


Figure 1. Flow chart of Simplified Lagrangian Receding Horizon heuristic

The basic operation of the SLRH is diagrammed in the flow chart of Figure 1. The algorithm operates on a clock-driven basis – i.e., the heuristic is executed at specified time intervals as opposed to whenever a machine becomes available. In this study, the algorithm was executed at timesteps of $\underline{\Delta T}$ clock cycles (whose value was determined by experiment, as described in Section VII), where each clock cycle represented 0.1 seconds in the simulation.

At each specified time interval, the SLRH algorithm cycled through each machine in the simulation. For each machine, the algorithm first checked to see if it was available at this clock cycle. If the machine was available, then the algorithm proceeded to attempt to map a subtask onto it. If the machine was not available, then the algorithm moved on to the next machine and checked availability on it. The machines were checked in simple numerical order. If no machine was available at this clock cycle, the simulation clock was incremented by the specified time interval and the algorithm started again.

In the simulation, the complete pass through all machines was accomplished without introducing any unnecessary machine idle time. Thus, a timestep of ΔT=1 clock cycle was permitted, although use of such a value resulted in lengthy heuristic execution times. The execution time of the heuristic in a real-time field application, however, could lead to significantly larger minimum ΔT values. Thus, the heuristic execution time is of considerable importance. Fortunately, the SLRH algorithm is amenable to a parallel hardware implementation. This option was not considered in this initial study, but remains the subject of continued investigation.

If the machine was available at this clock cycle, the algorithm proceeded to collect a "pool" of candidate subtasks, denoted by $\underline{U}$. The entire set of unmapped subtasks was reviewed to determine which ones can "feasibly" be mapped on the machine at this iteration. Feasibility for each subtask was defined by ensuring that it met two conditions: (a) all of its parent subtasks were previously mapped, and (b) adequate energy remained on the target machine for that subtask to execute at the secondary version level ***and*** communicate all the resulting data items to wherever they might need to go.

Unfortunately, the exact value of this last quantity depended upon the mapping of the subtask's children. The energy required to communicate data from one machine to another depends upon the two machines' respective bandwidths. In addition, subtasks mapped to the same machine required no energy to communicate between them.

Because the subtask's children could not have been mapped prior to this time, the energy that would be required to communicate to them cannot be accurately known. The algorithm resolved this uncertainty through a worst-case approach: the energy required to communicate to each subtask child was calculated assuming the child process had been mapped to the lowest bandwidth connection – and hence, would consume the maximum amount of communication energy from the subtask's machine.

Using this approximation and the execution time for the subtask's secondary version on the target machine, the total required energy was compared to the available energy level. If the energy available was not adequate, the proposed subtask mapping was declared "infeasible" and not included in the resulting candidate pool. Otherwise, the subtask was considered "feasible" and included in U.

This was clearly a conservative approach that potentially excluded a number of acceptable mappings. However, in this study, the communications energy proved to be a negligible factor in the calculations. Thus, the use of the worst-case communications energy was not found to significantly affect the mapping process.

The decision to map the primary versus secondary version was made at a later point – this step simply ensured that at least the secondary version could be executed on the target machine, thus saving the algorithm from expending effort on subtasks that would eventually have to be rejected. If the pool of candidate subtasks was empty (i.e., none of the remaining subtasks could be feasibly mapped on the target machine), the algorithm proceeded on to the next machine.

If the candidate pool U was not empty, the algorithm evaluated each subtask in the pool in terms of the impact it would have on a global objective function. This was done for both versions of each subtask. The algorithm then compared the results of the objective function evaluation for the two versions, and selected the version that maximized the value of the objective function. The selected version was retained in the pool – the other version was no longer considered during this iteration of the program.

The pool of candidate subtasks was subsequently ordered according to the resulting objective function from maximum to minimum value. Proceeding through the ordered pool of candidate subtasks, the program determined the earliest possible starting time for the subtask given precedence and communication requirements. No action was allowed to be scheduled earlier than the current clock cycle (i.e., the program would not allow the scheduler to look backward in time). The first subtask that was capable of being started within the specified time horizon $\underline{H}$ was selected for mapping.

Given the selected subtask and the corresponding specified version, the algorithm assigned the subtask and version to the target machine and scheduled all incoming communications. Finally, the algorithm updated the energy levels (including energy used for communications and subtask execution) of all machines, and stored a historical record of all critical parameters for later analysis.

Once a mapping to the target machine had been completed, the algorithm subsequently checked to see if all

subtasks had been successfully mapped, and terminated if this had been accomplished. If some subtasks were still unmapped, the algorithm continued to loop on the current machine cycle until all machines had been checked. Once this had been accomplished, the algorithm incremented the system clock by the specified time step, checked to see if the time constraint τ had been reached, and then restarted the previously detailed loop.

Of primary importance to the performance of the SLRH algorithm is the selection of an appropriate objective function. In the Lagrangian approach, the objective function is formed by combining the overall objective (in this study, maximizing the number of primary version subtasks that can be executed) with the specified system constraints on total system energy and application execution time, *AET*, defined as the time at which the last subtask is completed. Thus, the global objective function used for this study can be expressed as a weighted sum of terms involving the total number of primary version subtasks mapped, the energy consumed by the execution of all mapped subtask/version pairs, and the *AET* of the resulting solution. Using α, β, and γ as the weights, defining the total system energy (TSE) as

$$TSE = \sum_{j=0}^{|M|-1} B_p(j)$$

and denoting the total energy consumed by the mapping (TEC) as

$$TEC = \sum_{j=0}^{|M|-1} EC(j)$$

where *EC(j)* represents the energy consumed on machine j by executing all subtask/version pairs assigned to that machine (including all scheduled communications), the global objective function can be written as:

$$ObjFn(\alpha,\beta,\gamma) = \alpha\frac{T_{100}}{|T|} - \beta\frac{TEC}{TSE} + \gamma\frac{AET}{\tau}$$

Although only two weights are actually required, three weights were used in the study to allow easy investigation of system performance in the absence of any of the three terms.

This form of the objective function reflects the heuristic's focus on maximizing the objective function value. Larger values of T_{100} are rewarded, and larger consumption of energy penalized. Use of a negative sign on this term caused the heuristic to produce very short *AET* solutions, but with correspondingly lower T_{100} values. Given the stated objective for this study of maximizing T_{100}, this was considered an undesirable trade-off. Hence, the positive sign on the final term was selected to encourage use of all of the available time within the *AET*'s specified time constraint.

Each term of the objective function has been normalized to the [0,1] range. By constraining each of the weights to that range, and requiring that $\alpha+\beta+\gamma=1$, the objective function was confined to the same [0,1] range. Note that the "hard" boundary conditions on total system energy and application execution time are now expressed as a "soft" bias in the objective function, thus making it possible for solutions formed by this approach to actually violate a system constraint. In prior work, each solution was checked for constraint violation after the mapping was completed. While this approach is acceptable for the offline situations described in that work, the dynamic nature of the *ad hoc* grid problem precludes it. The solutions in the dynamic case must therefore be checked for constraint violation on an ongoing basis.

Thus, the feasibility check performed in establishing the pool of candidate subtasks *U* becomes a critical function. However, this check only verified compliance with the energy constraint. Verifying compliance with the specified constraint on *AET* was accomplished by adjustment of the objective function parameters. All mappings whose *AET* exceeded the specified τ were rejected, and their (α, β) values adjusted until the *AET* was brought into compliance.

V. Heuristics

Three variations of the SLRH heuristic were evaluated. In addition, a common static heuristic (Max-max) was used to provide a baseline for comparison. The heuristics included:

SLRH-1: The *Simplified Lagrangian Receding Horizon Heuristic – Variation 1* (SLRH-1) served as the baseline variation. It operated identically to the previous description.

SLRH-2: The *Simplified Lagrangian Receding Horizon Heuristic – Variation 2* (SLRH-2) followed the SLRH algorithm with a single major distinction. Whereas in SLRH-1 each machine was only assigned a single subtask/version pair at a given timestep, SLRH-2 continued to assign subtask/version pairs to a machine at a given timestep until either all pairs in the candidate pool *U* had been assigned or no additional pairs could be started within the time horizon. Therefore, subtask/version pairs would continue to be assigned to machine 1 (for example) before a new candidate pool was created and any of its subtask/version pairs were assigned to machine 2. Once all machines had been processed in this fashion, the timestep was incremented and the heuristic continued as in SLRH-1.

SLRH-3: The *Simplified Lagrangian Receding Horizon Heuristic – Variation 3* (SLRH-3) extended the SLRH-2 method by recreating and re-evaluating *U* after each subtask/version pair assignment. Thus, as a subtask was mapped, the heuristic immediately added that subtask's children to *U* (assuming all other requirements were met), re-evaluated all subtask/version pairs in *U*, and continued mapping on the same machine.

Max-Max: The *Max-Max* static heuristic used to provide a baseline for evaluating performance of the dynamic heuristics against this problem was based on the general "Min-Min" approach described in [IbK77]. The heuristic utilized the same objective function as the SLRH heuristics, but did not involve a receding horizon element. The heuristic began by creating a candidate pool *U* of "feasible" subtask/version pairs. However, for the Max-Max heuristic, the definition of "feasible" was slightly modified from that used in the SLRH heuristics. While the requirement that all parent subtasks be previously mapped was retained, the energy requirement was modified to separately consider each subtask version. Thus, the "feasibility" of both the primary and secondary versions were independently assessed. It was therefore possible for *U* to simultaneously include both versions of the same subtask.

The Max-Max heuristic subsequently selected from within the complete set *U* for each machine that subtask/version pair that provided the maximum increase in the objective function. From this subset, the subtask/version/machine triplet that provided the maximum increase in the objective function was selected and assigned to the corresponding machine.

The heuristic allowed a mappable triplet to be scheduled for a time prior to the target machine's availability time if a sufficiently large "hole" in the existing schedule could be found that complied with precedence constraints. This process continued until all subtasks were mapped.

VI. Upper Bound Calculation

Calculating an upper bound on the number of primary version subtasks that could be executed on a given set of machines requires consideration of the imposed limits on both time and energy. An estimate of the upper bound was computed for this problem using the concept of "equivalent computing cycles". In this method, each machine contributes a number of clock cycles to the system that is determined by the overall time limit for the scenario, adjusted to reflect the speed of that machine relative to a reference machine. Thus, the method creates a "pool" of equivalent clock cycles for the system that represents the total computing resources available to the system within the time limit constraint.

As the choice of reference machine was arbitrary, the approach used in this effort selected the 0th machine. The first step in the upper bound calculation scans through all subtasks and calculates the ratio of each subtask's execution time on each machine to that subtask's execution time on the reference machine. The method then finds the minimum ratio (MR) across all subtasks for each machine, given by:

$$MR(j) = \min_{i=0}^{|T|-1} \frac{ETC(i,j)}{ETC(i,0)}$$

The average minimum ratio and associated standard deviation for each machine, computed across all ten ETC matrices for each case, is shown in Table 3 below.

Table 3. Average Minimum Relative Speed

Case	"Fast" Machine 1	"Slow" Machine 1	"Slow" Machine 2
A	0.28 (0.03)	1.65 (0.18)	1.74 (0.3)
B	0.26 (0.03)	1.55 (0.32)	
C		1.63 (0.42)	1.59 (0.33)

The minimum ratio is a measure of the absolute minimum number of clock cycles on machine *j* that are required to complete the same amount of work as on the reference machine. Each machine thus contributes a number of "equivalent cycles" to the system's pool that is equal to the overall time limit divided by the MR for that machine. For example, a machine whose minimum ratio was 2.0 (indicating that the time required for any subtask to complete on it was at least twice as long as on the reference machine) would contribute a number of equivalent cycles to the pool equal to the time limit τ divided by 2. This represents the best-case situation, thus ensuring that the method provides an upper bound on the number of subtasks that could be executed with their primary version. Thus, the total available equivalent computing cycles (TECC) within the system is given by:

$$TECC = \sum_{j=0}^{|M|-1} \frac{\tau}{MR(j)}$$

Given this pool of computing cycles and the previously defined total system energy (TSE), the upper bound calculation proceeded by searching the primary version of all previously unused subtasks across all machines to find the subtask-machine pair that consumed the minimum amount of energy. The number of equivalent cycles required to execute the primary version of this subtask

was computed by dividing its execution time on that machine by the machine's MR. If the system had both enough total equivalent computing cycles and enough total system energy to complete the subtask, then the number of primary version subtasks that can be executed was incremented by one and the energy and equivalent cycles associated with that subtask were subtracted from the system's resources. This process continued until either insufficient energy or equivalent cycles remained to complete the primary version of the selected subtask.

The results of the upper bound calculation for the three simulation cases are shown in Table 4 below for each of the ten ETC matrices. In general, the upper bound calculation revealed that the system as a whole had adequate energy and compute cycles to complete primary versions of all subtasks for Case A and Case B, although the distribution of those resources across the individual machines precluded such a solution. This was not the situation in Case C, where the lack of sufficient compute cycles was the primary limit on the upper bound.

Table 4. Results of upper bound calculation

ETC	Case A 2 fast, 2 slow	Case B 2 fast, 1 slow	Case C 1 fast, 2 slow
0	1024	1024	817
1	1024	1024	862
2	1024	1024	816
3	1024	1024	654
4	1024	1024	900
5	1024	1024	772
6	1024	1013	733
7	1024	1024	804
8	1024	1024	756
9	1024	1024	724

VII. Experimental Results

Establishing a value for the ΔT parameter within the SLRH variants was accomplished by considering the parameter's impact on both T_{100} and execution time. Large values of ΔT resulted in fewer iterations of the heuristic and potentially large gaps of unused computational cycles, thus causing a reduction in the number of primary version subtasks that could be mapped within the time constraint. Small values of ΔT, however, resulted in a large number of heuristic iterations that provided no successful mapping of a subtask.

These trends are reflected in the results shown in Figure 2 for SLRH-1 operating on ETC 0 for two DAGs. The top two lines in the figure illustrate the impact of ΔT on T_{100} for the SLRH-1 heuristic operating on ETC 0 in Case A for each of the two DAGs. The figure shows T_{100} to be relatively insensitive to ΔT for mid-range values. The lower two lines, however, illustrate the impact on heuristic execution time under the same conditions. As the figure shows, the heuristic displays a strong dependence between heuristic execution time and ΔT for small values.

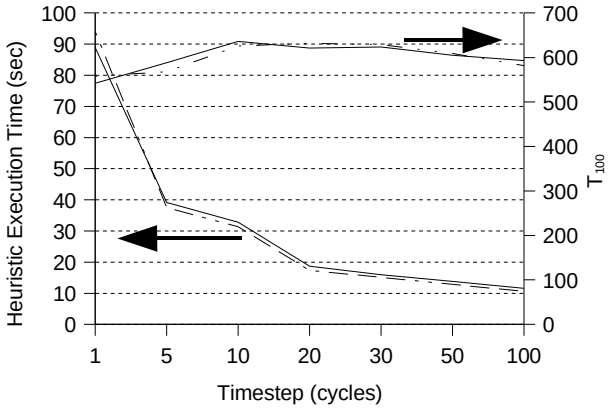


Figure 2. Impact of ΔT parameter on SLRH-1

No significant variation in behavior with respect to ΔT was found between the SLRH variants or the various ETC/DAG combinations.

Similar analyses were performed for the SLRH time horizon, H. Large values of H allowed the heuristic to map subtasks for distant starting times, thereby reducing the heuristic's flexibility for scheduling subsequent subtasks. However, for this study, the impact of H on both T_{100} and execution time was found to be negligible. Based on the results of these analyses, values of 10 clock cycles for ΔT and 100 clock cycles for H were used in the remainder of the study.

A primary objective of the study was to determine the sensitivity of the Lagrangian Receding Horizon concept to the objective function weights. A relatively insensitive heuristic might be able to operate in an *ad hoc* grid's dynamic environment without incorporating a method for online adjustment of the weights. More sensitive heuristics, however, would require such methods to ensure adequate levels of performance.

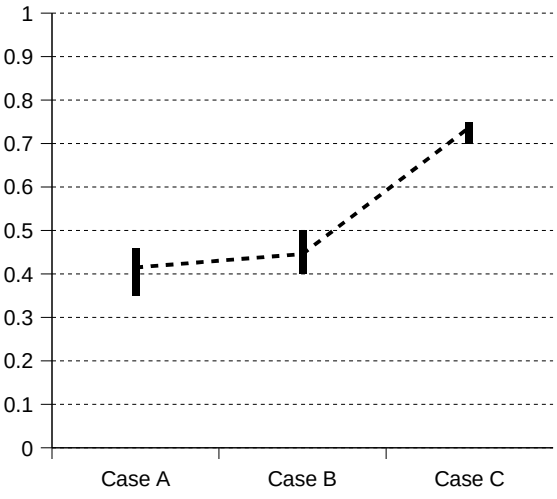
The sensitivity of the heuristics to the objective function weights was investigated by first independently varying the α and β values across their [0,1] range in steps of 0.1 until a general range was found that produced the best T_{100} performance, subject to the energy and time constraints. In addition, the heuristic was required to successfully map all 1024 subtasks within both the specified energy and time constraints for that (α,β) combination to be included in the study. The values were then varied by

0.02 across this smaller range until an optimal performance point was determined.

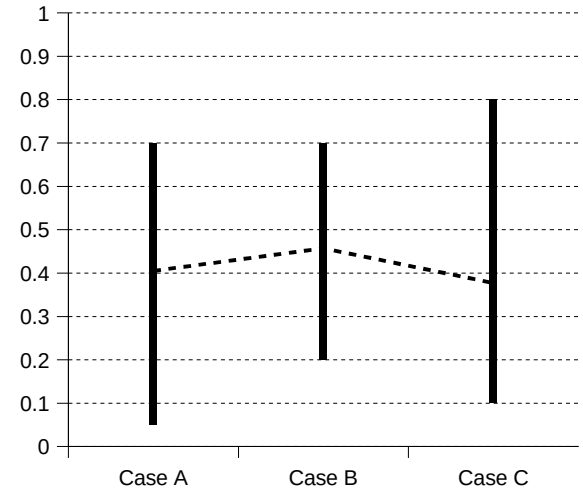
This optimality analysis was performed using each heuristic for each ETC/DAG combination in all three cases. The result of these experiments was a set of (α,β) pairs for each case, each element of the set containing the (α,β) values that provided the best performance with respect to T_{100} for a particular ETC/DAG combination. Each set was subsequently analyzed to find the average, minimum, and maximum values of α and β.

The results of the analysis are shown in Figure 3, below. The dashed lines connect the average value of the respective parameter resulting from the analysis. The vertical bars in each graph show the range from the minimum to maximum value of the respective parameter for each case. The set of optimal (α,β) pairs for each case were essentially identical for the SLRH-1 and SLRH-3 variants – thus, their respective average, minimum, and maximum values overlay each other on the graph. The SLRH-2 variant, however, was found to rarely produce a successful mapping of all 1024 subtasks within the time and energy constraints regardless of the choice of α and β. This variant was subsequently dropped from further consideration and is not included in the graph.

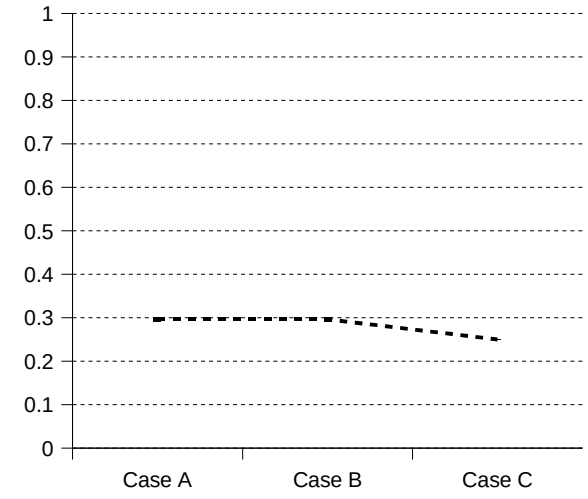
The SLRH heuristic demonstrates a small degree of variation in α across the various ETC/DAG combinations. As Figure 3(a) shows, the optimal α value for SLRH performance varied somewhat across the three cases. While Cases A and B were relatively similar in both the value and range of α, Case C differed significantly in both areas. Not only did the optimal value of α change by over



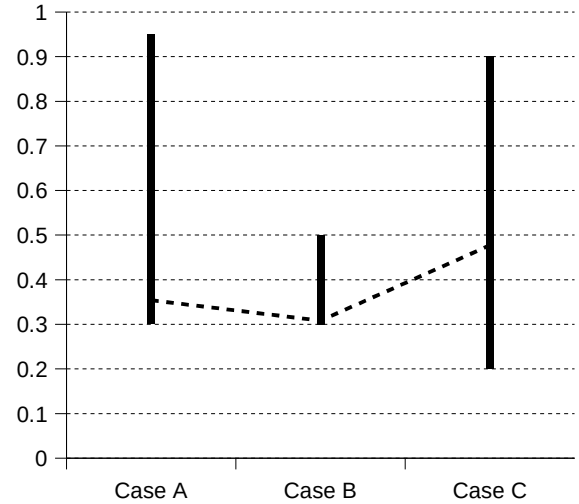
(a) α variability for SLRH heuristics



(c) α variability for Max-Max heuristic



(b) β variability for SLRH heuristics



(d) β variability for Max-Max heuristic

Figure 3. Comparison of (α,β) variability in SLRH and Max-Max heuristics.

50%, but the range of values that provided optimal performance shrank a corresponding amount.

The β parameter for the SLRH heuristic, shown in Figured 3(b), exhibited even less variation than was found for the α parameter, with the heuristic optimizing its performance at the same value for every ETC/DAG combination in all three cases. There was a slight change in value, however, between Cases A and B, and Case C. Off-optimal operation of the heuristic in this last case by using the same value of β as in Cases A and B resulted in a loss of performance of less than 5%.

In contrast, the Max-Max heuristic exhibited a very wide range of optimal α and β values, as shown in Figures 3(c) and 3(d). No direct correlation was found between the optimal (α,β) combination and either the ETC or DAG involved in the simulation. An exhaustive search was therefore required to identify the optimal (α,β) pair for Max-Max in each scenario.

Based on these results, it appears that the SLRH heuristic will require a method for dynamically adjusting at least the α parameter in response to changes in the *ad hoc* grid environment.

The results shown in the remainder of this paper utilize optimal values for α and β for each ETC/DAG combination. The performance of the three remaining heuristics (SLRH-1, SLRH-3, and Max-Max) was evaluated in each of the three cases. Each result reported is the average of the outcomes from all 100 ETC/DAG combinations.

Figure 4 compares the performance of the heuristics against the primary objective – maximizing the number of primary version subtasks that could be executed. As the figure shows, the SLRH-1 variant provided roughly equivalent performance to that of the baseline provided by the static Max-Max heuristic in Case A, both of which significantly outperformed the SLRH-3 variant.

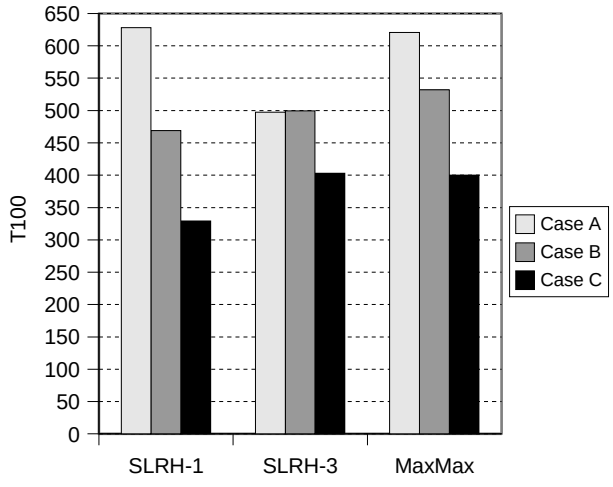


Figure 4. Comparison of heuristic performance based on number of primary versions mapped

As expected, both the SLRH-1 and Max-Max heuristics displayed a marked drop-off in performance as machines are "lost" to the system. The SLRH-1 heuristic fell off faster than the static Max-Max baseline, however, indicative of the dynamic SLRH heuristic's more limited knowledge of upcoming subtasks. The SLRH-3 heuristic maintained its performance when losing one "slow" machine. This was attributed to its poor performance when all machines were present, as opposed to being inherently more resilient than the other heuristics.

Performance of all three heuristics relative to the calculated upper bound is shown in Figure 5 below. The SLRH-1 performed well for Case A, achieving mappings that averaged T_{100} values better than 60% of the upper bound and were slightly better than the static Max-Max baseline. Both of these heuristics suffered a significant drop in performance when a machine was removed from the system, with the Max-Max baseline heuristic showing itself to be slightly less sensitive to the loss of a machine. As the figure shows, however, the performance impact relative to the upper bound was relatively independent of the removed machine's type.

Performance of the SLRH-3 in Case A was significantly poorer than the other heuristics. However, the heuristic proved to be fairly insensitive (relative to the upper bound) to the loss of machines, regardless of type.

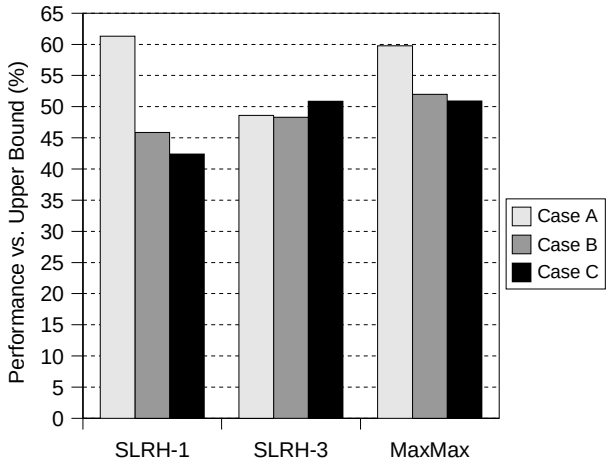


Figure 5. Heuristic performance vs. calculated upper bound

The average execution time for each of the heuristics to successfully map all 1024 subtasks is shown in Figure 6. Timing experiments were conducted using the Python 2.3.3 scripting language on a non-dedicated 2.1GHz dual Xeon processor computer with 2GB RAM and 512KB L2-cache, running RedHat Linux version 9.0. Although

the code was carefully written, no significant attempt to optimize the code for execution time was made. The values shown represent the average heuristic execution time across all ETC/DAG combinations for each case, with the optimal values of α and β used for each ETC/DAG combination.

Execution times of the Max-Max baseline heuristic were relatively constant across the three cases, as expected from the static nature of that heuristic. The SLRH-3 variant displayed significant sensitivity to loss of machines as this dictated consideration of a larger number of alternative mappings at each iteration. In contrast, SLRH-1 showed only a small increase in execution time when losing a "slow" machine, and actually reduced its execution time significantly when losing a "fast" machine. This was attributed to the heuristic's ability to efficiently utilize mappings of subtasks' secondary versions to take advantage of the energy available on the "slow" machines without violating the specified constraint on *AET*.

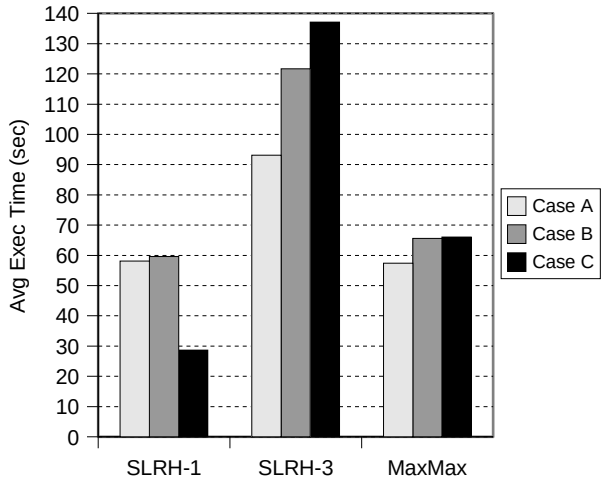


Figure 6. Comparison of average execution time of heuristics

Significant speed improvements over the performance shown in Figure 6 may be possible using compiled languages and through further optimization of the code. In addition, tests on SLRH-1 indicate that execution speed improvements of 15-20% are achievable through elimination of bookkeeping associated with instrumenting the experiment and not associated directly with the algorithm.

For dynamic scenarios, comparing the relative value of the heuristics requires definition of a metric that reflects both the value of the mapping produced by the heuristic, and the execution time of the heuristic itself in producing that mapping. A simple metric based on the ratio of T_{100} to the heuristic's execution time is shown in Figure 7.

Focused solely on the performance of the heuristic in terms of the primary objective, the metric favorably reflects the execution speed of the SLRH-1 variant relative to its SLRH-3 cousin. Both SLRH-1 and Max-Max continue to exhibit relatively equal performance for Cases A and B. However, the heuristics show a marked departure from that parity when confronted by the loss of a "slow" machine. The dynamic SLRH-1 significantly outperformed the static Max-Max heuristic in this scenario, primarily due to its faster execution time.

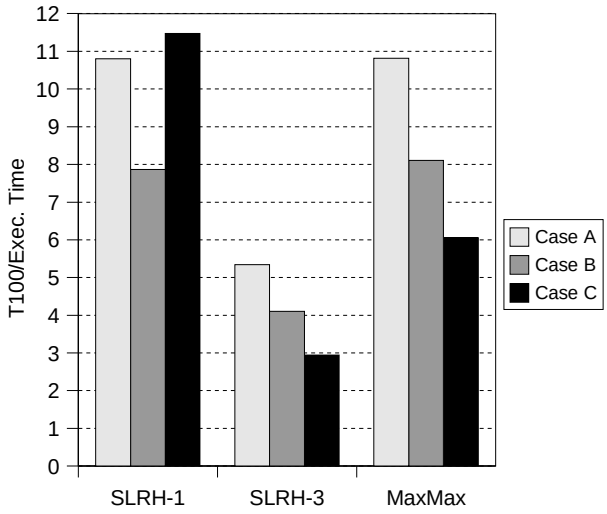


Figure 7. Simple metric of performance per unit of heuristic execution time

VIII. Summary

This paper presented one potential method for efficiently and robustly managing both computational and communication resources in a dynamic, unpredictable environment. Formed by combining Lagrangian-based resource management methods with the receding horizon concept from the control community, the Simplified Lagrangian Receding Horizon heuristic's performance was evaluated against several cases reflecting the loss of machines from the system.

With all machines present, the SLRH heuristic performed comparably to the baseline provided by a static Max-Max heuristic in terms of T_{100}. The heuristic displayed a significant sensitivity to the loss of machines, but performed very well when measured against its execution time.

The need for on-the-fly adjustment of the Lagrangian parameters in Lagrangian receding horizon heuristics was also demonstrated. When confronted with changes in the system's available resources, the heuristic was particularly sensitivity to the T_{100} multiplier, thereby indicating that this value requires adjustment whenever the system environment changes. In contrast, the constraint multipliers (β

and γ) may either require less frequent adjustment or perhaps be held constant.

The static nature of the Max-Max heuristic used to provide a baseline for evaluating the performance of the dynamic SLRH heuristics precludes its use in *ad hoc* grid environments. However, the comparable performance provided by the SLRH-1 heuristic in this initial study indicates a significant potential to efficiently manage both computational and communication resources in this dynamic, unpredictable environment – despite the lack of full knowledge of tasks that may be assigned to the grid.

Acknowledgments: The authors thank Shoukat Ali and Sameer Shivle for their assistance in generating the test data. This research was supported in part by the Colorado State University George T. Abell Endowment.

References

[AlS00] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Modeling Task Execution Time Behavior in Heterogeneous Computing Systems," *Tamkang Journal of Science and Engineering*, Special Tamkang University 50th Anniversary Issue, vol. 3, no. 3, pp. 195-207, Nov. 2000, (invited).

[BrH75] A.E. Bryson and Y-C Ho, *Applied Optimal Control*, Hemisphere Publishing Corporation, New York, 1975.

[CaS03] R.H. Castain and W.W. Saylor, "Static Mapping for Ad Hoc Grid Subtasks with Dependencies and Time and Energy Constraints using Lagrangian Relaxation Neural Networks", EE 680 Project Report, Colorado State University, Ft. Collins, CO, Apr 2003 (unpublished).

[Go50] H. Goldstein, *Classical Mechanics*, Addison-Wesley, Massachusetts, 1950.

[IbK77] O.H. Ibarra and C.E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors", *Journal of the ACM*, vol. 24, no. 2, Apr. 1977, pp. 280-289.

[LeM67] E.B. Lee and L. Markus, *Foundations of Optimal Control Theory*, Wiley, New York, 1967.

[LuH93] P.B. Luh and D.J. Hoitomt, "Scheduling of manufacturing systems using the Lagrangian relaxation technique", *IEEE Transactions on Automation and Control*, vol. 38, no. 7, 1993, pp. 1066-1079.

[LuZ00] P.B. Luh, Y.W. Zhao, and L.S. Thakur, "Lagrangian relaxation neural networks for job shop scheduling", *IEEE Trans. on Robotics and Automation*, vol. 16, no. 1, Feb. 2000, pp. 78-88.

[MaM03] D. Marinescu, G. Marinescu, Y. Ji, L. Boloni, and H.J. Siegel, "Ad hoc grids: Communication and computing in a power constrained environment", *Workshop on Energy-Efficient Wireless Communications and Networks 2003 (EWCN 2003)*, cosponsors: IEEE Computer Society and IEEE Communications Society, in the proceedings of the 22nd International Performance, Computing, and Communications Conference (IPCCC), Apr. 2003.

[PeS97] F.L. Pereira and J.B. de Sousa, "On the receding horizon hierarchical optimal control of manufacturing systems", *Journal of Intelligent Manufacturing*, vol. 8, 1997, pp. 425-433.

[ShC04] S. Shivle, R.H. Castain, H. J. Siegel, A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa1, J. Sridharan, P. Sugavanam, and J. Velazco, "Static mapping of subtasks in a heterogeneous ad hoc grid environment", to be presented at the *13th International Heterogeneous Computing Workshop*, cosponsors: IEEE Computer Society and Office of Naval Research, to appear in the proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS), Apr. 2004.

[WaS97] L. Wang, H.J. Siegel, V.P. Roychowdhury, and A.A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, vol. 47, no. 1, pp. 8-22, Nov. 25, 1997.

[WhS92] D.A. White and D.A. Sofge, ed., *Handbook of Intelligent Control*, Van Nostrand Reinhold, New York, 1992.

[YuH01] C-Y Yu and H-P Huang, " Priority-based tool capacity allocation in the foundry fab", *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, Seoul, Korea, pp. 1839-1844, May 21-26, 2001.

Biographies

Ralph H. Castain is currently serving as a Research Scientist within the Electrical and Computer Engineering Department at Colorado State University where he conducts research focused on robust resource management within distributed computing systems. In addition, he is leading the Colorado Grid Computing (COGrid) Initiative on behalf of Colorado State University, an effort that he founded in late 2002 to create a statewide grid computing system capable of meeting the needs of industry, government, and academia of all levels. Prior to joining the University, he spent eight years in industry leading technology initiatives, and eleven years at Los Alamos National Laboratory. While at Los Alamos, he served as Chief Scientist for Nonproliferation and Arms Control. His technical paper in the early 1990s on next generation methods for proliferation detection has served as the foundation for the U.S. government's nonproliferation research program for over ten years. He received his BS degree from Harvey Mudd College, and the MS, MSEE, and PhD degrees from Purdue University.

William W. Saylor is currently working as a consultant for the Department of Defense on several advanced

technology programs and is also in a graduate program at Colorado State University doing research on control issues for complex systems. He has spent the past eight years working in the defense and energy industries after twelve years at the Los Alamos National Laboratory, where he was a project leader for several aerospace and defense efforts. Prior to that he worked as a nuclear engineer in the energy industry and spent nine years in the U. S. Army. He received his BS degree from The United States Military Academy and an MS degree from MIT.

H. J. Siegel holds the endowed chair position of Abell Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU), where he is also a Professor of Computer Science. He is the Director of the CSU Information Science and Technology Center (ISTeC). ISTeC a university-wide organization for promoting, facilitating, and enhancing CSU's research, education, and outreach activities pertaining to the design and innovative application of computer, communication, and information systems. Prof. Siegel is a Fellow of the IEEE and a Fellow of the ACM. From 1976 to 2001, he was a professor in the School of Electrical and Computer Engineering at Purdue University. He received a B.S. degree in electrical engineering and a B.S. degree in management from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He has co-authored over 300 technical papers. His research interests include heterogeneous parallel and distributed computing, communication networks, parallel algorithms, parallel machine interconnection networks, and reconfigurable parallel computer systems. He was a Coeditor-in-Chief of the Journal of Parallel and Distributed Computing, and has been on the Editorial Boards of both the IEEE Transactions on Parallel and Distributed Systems and the IEEE Transactions on Computers. He was Program Chair/Co-Chair of three major international conferences, General Chair/Co-Chair of four international conferences, and Chair/Co-Chair of five workshops. He is currently on the Steering Committees of five continuing conferences/workshops. He is a member of the Eta Kappa Nu electrical engineering honor society, the Sigma Xi science honor society, and the Upsilon Pi Epsilon computing sciences honor society. *An up-to-date vita is available at www.engr.colostate.edu/~hj.*