Robust Resource Allocation of DAGs in a Heterogeneous Multicore System

Luis Diego Briceño[1], Jay Smith[1,3], Howard Jay Siegel[1,2],
Anthony A. Maciejewski[1], Paul Maxwell[1,4], Russ Wakefield[2],
Abdulla Al-Qawasmeh[1], Ron C. Chiang[1], and Jiayin Li[1]

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

[3]DigitalGlobe
Longmont,CO,80503

[4]United States Army

Email: ldbricen@colostate.edu, jtsmith@digitalglobe.com,
{hj, aam, paul.maxwell, russ.wakefield, abd, chilung, lijiayin}@colostate.edu

***Abstract*—In this study, we consider an environment composed of a heterogeneous cluster of multicore-based machines used to analyze satellite images. The workload involves large data sets, and is typically subject to deadline constraints. Multiple applications, each represented by a directed acyclic graph (DAG), are allocated to a dedicated heterogeneous distributed computing system. Each vertex in the DAG represents a task that needs to be executed and task execution times vary substantially across machines. The goal of this research is to assign applications to multicore-based parallel system in such a way that all applications complete before a common deadline, and their completion times are robust against uncertainties in execution times. We define a measure that quantifies robustness in this environment. We design, compare, and evaluate two resource allocation heuristics that attempt to maximize robustness.**

I. Introduction

We consider a heterogeneous computing (**HC**) system based on multicore chips used to analyze satellite data. The data processing applications used in the analysis typically require computation on large data sets, and their execution may be subject to a completion deadline. Multiple applications, each represented as a directed acyclic graph (**DAG**) of tasks, are to be assigned to an HC system for execution. The goal of this study is to assign tasks to processors in such a way that all applications complete before a common deadline, and the application completion times are robust against uncertainties in task execution times. We define a measure of robustness in this context, and we design, compare, and evaluate two resource allocation heuristics that attempt to maximize robustness.

The simulation environment used to compare and evaluate these heuristics is motivated by similar systems in use at DigitalGlobe[±] and the National Center for Atmospheric Research (**NCAR**[⋆]). In these systems, data from a satellite is received and distributed to storage units in the satellite data processing HC system, where there are: (a) heterogeneous hard drive (**HD**) access rates, (b) different computational capabilities across compute nodes, and (c) different data set sizes. This satellite data processing system has the following characteristics: (a) the initial allocation of satellite data to HDs is determined by the heuristic, (b) there is limited RAM available at each machine, (c) data items have to be explicitly staged and removed from RAM, (d) some tasks can be executed using parallelization, and (e) transfer times between HD and RAM must be taken into account. The simulation environment models this HC system and the applications.

Each application only requires a subset of the total collection of data that is downloaded from the satellite. Resource allocation in this environment requires both selecting a location within the system to store each satellite data item and mapping tasks to compute nodes for execution. All applications and their required satellite data items are known prior to the satellite collecting the data, so this is an instance of a static resource allocation problem [1]. The general mapping problem is NP-complete [6], [8], [11]; therefore, heuristics are required

This research was supported by the NSF under grants CNS-0615170 and CNS-0905399, and by the Colorado State University George T. Abell Endowment.

± http://www.DigitalGlobe.com/
⋆ http://www.ncar.ucar.edu/

978-1-4244-6534-7/10/$26.00 ©2010 IEEE

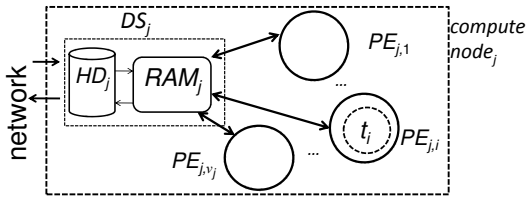

Fig. 1. In this figure, the composition of compute node j is illustrated.

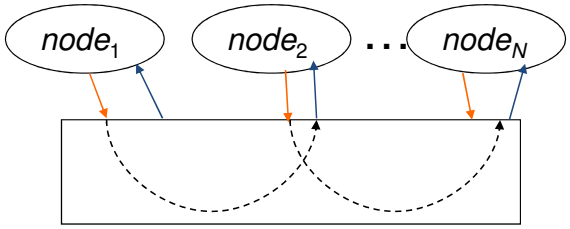

Fig. 2. In this figure, an illustration of HiPPI network is shown.

to obtain a near-optimal allocation in a reasonable time.

Our contributions are: (a) a model and simulation of a complex multicore-based data processing environment that executes data intensive applications, (b) a robustness metric for this environment, and (c) resource allocation heuristics to maximize robustness using this metric.

In the next section, we will describe the problem statement. Two new heuristics are defined in Section III. The related work is discussed in Section IV. Section V and VI provide the results and conclusions, respectively.

II. Problem Statement

A. System Model

The goal of resource allocations in this environment is to complete all applications before a common deadline ($\boldsymbol{\Delta}$). Let $\boldsymbol{app_k}$ be the k^{th} application. Each app_k is divided into $\boldsymbol{T_k}$ tasks, represented by a DAG. In the DAG, entry tasks require only satellite data and exit tasks produce final results that must be stored on an HD.

Tasks may require both satellite data sets, denoted $\boldsymbol{SD_i}$, and data sets produced by other tasks, denoted $\boldsymbol{TD_i}$. Within a compute node α, a data set can be located in either RAM or HD. The location of a data set (TD or SD) within a compute node α is denoted $\boldsymbol{loc_\alpha}$, i.e., $loc_\alpha \in \{RAM_\alpha, HD_\alpha\}$.

This HC system is composed of $\boldsymbol{N}$ compute nodes, where each compute node j has dedicated storage ($\boldsymbol{DS_j}$), composed of RAM_j and HD_j. Each compute node also has one to eight processing elements (**PEs**), and each PE may only execute one task at a time (i.e., no multi-tasking). Each node is comprised of a collection of multicore chips. The PEs within a compute node are homogeneous, but across compute nodes PEs are assumed to be heterogeneous. Each compute node j has $\boldsymbol{\nu_j}$ PEs where the x^{th} PE is denoted $\boldsymbol{PE_{j,x}}(1 \leq x \leq \nu_j)$, and the total number of PEs across all compute nodes is $\boldsymbol{M}$ ($M = \sum_{j=1}^{N} \nu_j$). The composition of a compute node j is shown in Figure 1.

In each compute node, the RAM storage space is limited and may be unable to store all needed data sets simultaneously; however, each HD is assumed to be large enough to store any SDs and TDs assigned to it. If a TD in RAM is to be released, but is required later as an input to a task, then it must be copied to the HD. Because all SDs are initially stored on HDs, SDs do not need to be saved to the HD before being overwritten in RAM.

All the input data sets required by a task must be in local RAM before the task can start executing, and must remain in RAM until its execution is finished. The storage space in RAM for the output of a task must be locally reserved before it begins execution. In this system, all PEs on the same compute node share dedicated storage and network access. The time to access local RAM from a PE is assumed negligible; but HD access is not, and is different for a read or a write.

The network topology used for this study is a high performance parallel interface (**HiPPI**) crossbar switch (Figure 2). Each compute node may simultaneously transmit and receive one data set at a time, but may not broadcast. The transfer rate of data from one compute node to another depends on the data's location in both the source and the destination, i.e., RAM or HD.

Because there are two locations where data may be stored and two compute nodes involved in the transfer, there are four cases. In the first case, we wish to transfer data from RAM on a source compute node to RAM on a destination compute node. This transfer is only limited by network bandwidth (the same for all compute nodes) because the bandwidth to RAM is always greater. In the second case, the transfer rate of data from the HD on the source compute node to RAM on the destination compute node is limited by the smaller of the network bandwidth and the read bandwidth of the source HD. In the third case, the transfer rate of data from RAM on a source compute node to HD on a destination compute node is limited to the smaller of the network bandwidth and the write bandwidth of the destination HD. In the fourth case, the transfer rate of data from a HD on the source compute node to the HD on the destination compute node is limited by the smaller of the network bandwidth, the read bandwidth from the source HD and the write bandwidth to the HD on the destination compute node.

For each task t_i, we assume that an estimated time

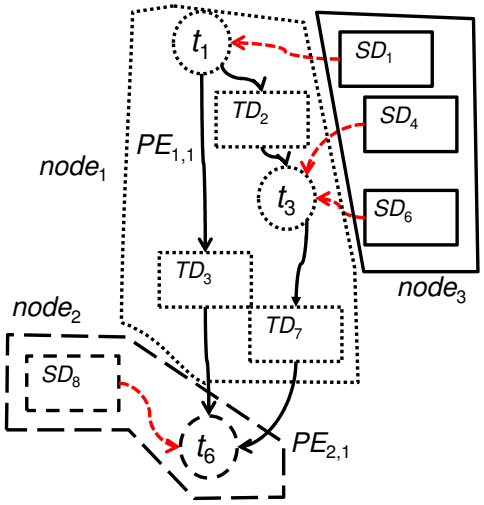

Fig. 3. This figure shows a diagram of a DAG. The PE shown in compute node 1 ($PE_{1,1}$) is executing t_1 that requires SD_1 from compute node 3. In this case, TD_3 and TD_7 on compute node 1 need to be transmitted to compute node 2 for t_6. The result of t_6 must be stored in an HD of the system, and the time to store the result must be considered when calculating the makespan.

to compute on each compute node j has been provided, denoted $\boldsymbol{ETC(i,j)}$, possibly determined from past task execution times, which is a common assumption (e.g., [5], [7], [9], [12], [13], [19], [22]). The goal of this study is to assign tasks to PEs so that unexpected increases in the estimated task computation times do not cause the total time required to complete all applications (makespan) to exceed Δ. See Figure 3 for an example of a resource allocation of a DAG.

A subset of the tasks are designed to be decomposable for parallel execution within a single compute node. Each decomposable task is assigned a **divisor** value that indicates how amenable the task is to parallel processing, $\boldsymbol{ETC_{parallel}(i,j)} = \frac{ETC(i,j)}{divisor}$. Tasks are grouped into good parallel tasks and poor parallel tasks. The divisor values we use in the simulations are shown in Table I.

B. Robustness

A resource allocation is robust if it meets a given performance criterion and is able to maintain this performance despite unexpected perturbations [4] (e.g., [16], [20]). To quantitatively compare robustness among different possible resource allocations, three questions about robustness must be answered [3]: *(1) What behavior of the system makes it robust?* Our system is robust if all applications complete before a common deadline Δ. *(2) What uncertainties is the system robust against?* The uncertainty is the relationship between the estimated execution time of each task and the actual data dependent execution time of each task. *(3) Quantitatively, exactly how robust is the system?* The robustness of a given resource allocation is the smallest common percentage increase ($\boldsymbol{\rho}$) for all task execution times that causes the makespan to be equal to the deadline Δ. Thus, the robustness metric ρ is maximized for this study. In general, minimizing makespan will not maximize robustness. An example of a resource allocation is shown in Figure 4.

C. Performance Metric

Robustness cannot be calculated just by using the estimated makespan of a resource allocation because of the complexity introduced by the inter-compute node data transfers, i.e., robustness is different from just increasing the makespan by ρ.

In a real system, the execution times of all tasks will not be increased by the same percentage. However, ρ can be used as a suitable measure for robustness in this environment—it can be viewed as a worst-case guarantee.

The performance metric for this study is the robustness ρ. Due to the complexity of the environment, it is difficult to identify a closed-form expression for the robustness of a resource allocation. Thus, an iterative search procedure is used.

We define the makespan with a $\lambda\%$ increase in execution times as $\boldsymbol{makespan_{\lambda}}$. One procedure to calculate λ, such that $makespan_{\lambda} = \Delta$, is to multiply all the estimated task execution times by $1 + \lambda$. Then, using a binary search, the value of λ is found to the nearest percent. The starting upper value of λ for the binary search is an upper limit on λ ($\boldsymbol{UL_{\lambda}}$).

UL_{λ} for the binary search is calculated as follows. For each PE, sum the ETC values of the tasks assigned to that PE. Let μ be the maximum value of these sums among all PEs. The starting value for the binary search is:

$$UL_{\lambda} = \frac{\Delta}{\mu} - 1. \quad (1)$$

The binary search will go between 1% and UL_{λ}. For each iteration of the binary search, we calculate the makespan (including communication) until you find the value of λ (to the nearest percent) gives $makespan_{\lambda} = \Delta$.

III. Heuristics

A. Dynamic Available Tasks Critical Path (DATCP)

The DATCP heuristic (based on the concept of Dynamic Critical Path (DCP) [15]) uses the idea that with an infinite number of processors an application cannot execute any faster than its critical path length. Without loss of generality, assume that all of the entry nodes

TABLE I
TABLE SHOWING THE DIVISOR FOR PARALLEL TASKS

types of parallelism	number of PEs in use 1	2	3	4	5	6	7	8
divisor for good parallel tasks	1	1.75	2.5	3.25	4	4.75	5.5	6.25
divisor for poor parallel tasks	1	1.5	2	2.5	3	3.5	4	4.5

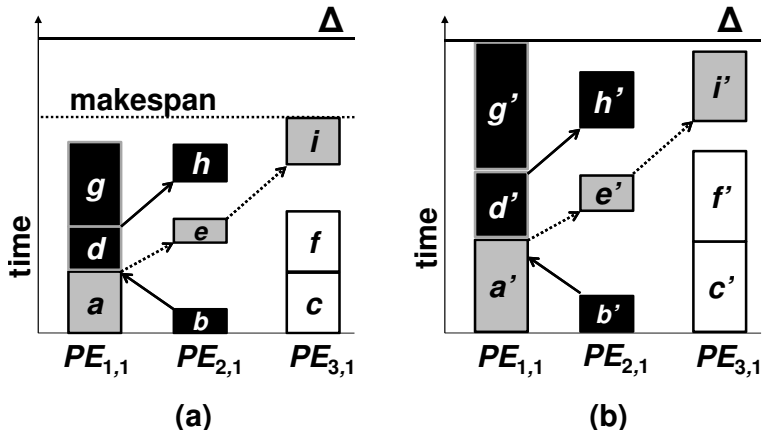

Fig. 4. In this figure, an example of a resource allocation is shown. Each of the execution times for the tasks in (a) is increased by 50% with the results shown in (b). Note that the communication times do not increase, and the makespan for (b) (equal to Δ) is much less than makespan in (a) increased by 50%. In (a), the makespan PE is $PE_{3,1}$. After all task execution times are increased by 50%, $PE_{3,1}$ is no longer the makespan PE. This example intuitively shows how the makespan is not a good measure of robustness in an environment with inter-task communication.

(1) Let $t_i = t_{exit}$
(2) Calculate SD estimated transfer time to t_i (size of SD divided by the average bandwidth time from HD).
(3) Calculate average task execution time ($AET(t_i)$).
(4) Calculate TD estimated transfer time to successor nodes (size of TD divided by network transfer bandwidth).
(5) Determine the maximum time from any successor (child) node to the t_{exit} ($\boldsymbol{max_{time}}$).
(6) The critical path value is the sum of TD and SD data transfer times, max_{time}, and $AET(t_i)$.
(7) Select another task (whose successors have a calculated critical path value) and go to step (2) until all tasks are processed.

Fig. 5. Procedure used to calculate the critical path of the DATCP heuristic.

of all application DAGs have a common predecessor that is the pseudo-vertex entry task $\boldsymbol{t_{entry}}$. Similarly, assume there is a unique exit node $\boldsymbol{t_{exit}}$. The calculation of the critical path is a recursive process that begins with t_{exit} and finishes at t_{entry}. Each task calculates its execution time to t_{exit} and then passes this time value to its predecessor tasks. The execution time for a task is calculated using the average values across all compute nodes. We define the average execution time of t_i as $\boldsymbol{AET(t_i)}$, and calculate it as follows:

$$AET(t_i) = \frac{\sum_{j=1}^{N} ETC(i,j) \cdot \nu_j}{M}. \quad (2)$$

To estimate the transfer times, two average bandwidths are used: (a) the **average bandwidth time from the HD** is the average read bandwidth of all compute nodes, and (b) the **network transfer bandwidth** is used to estimate the TD being sent across the network. The pseudo-code for the critical path calculation method is in Figure 5.

The next step of the heuristic determines the list of available tasks, which are entry tasks or tasks whose predecessors have been mapped. The available task with the maximum critical path time is then mapped to the compute node that maximizes the robustness value ρ. That task is then removed from the list of mappable tasks and the process is repeated until all tasks are mapped. The pseudo-code for this algorithm is listed

(1) Calculate the critical path for each application.
(2) Dynamically create a list of all tasks available for mapping.
(3) Determine the task with the longest critical path from the list of available tasks.
(4) For task t_i determined in (3),
  (a) For each PE k in each compute node j, calculate the robustness of $PE_{j,k}$ if t_i were mapped to node j (see Section III-A3 for details about SD placement).
  (b) Find the compute node j with the highest number of maximum-robustness-value PEs (it is possible for other PEs to have the same robustness value). This compute node is denoted $\boldsymbol{node_{max}}$.
  (c) Schedule communication transfer from predecessor tasks and satellite data transfer to $node_{max}$.
  (d) Map task to the PE (or PEs) with the maximum robustness (see Section III-A2 for details).
  (e) Remove task t_i from list.
(5) Repeat steps (2)–(4) until all tasks are mapped.

Fig. 6. DATCP heuristic procedure used to generate a resource allocation.

in Figure 6. If two possible assignments have the same robustness, then DATCP selects a PE within the compute node where the most PEs have the same robustness.

1) Memory Management: The heuristic schedules data set transfers as required by the resource allocation. When a task is considered for scheduling, the available space in RAM is determined to decide if the task and the input data it requires can be stored in RAM immediately, i.e., is there enough space in RAM. If there is not enough space, the heuristic checks when the task's input data sets can be moved into memory and schedules it to start execution at that time.

If a data set is transferred from a compute node i to a compute node j and there is space available in RAM_j, then it is transfered directly into RAM_j. If space is not available in RAM_j, then the data is moved to HD_j. When evaluating which data sets can be removed from RAM, the algorithm selects data sets that are not currently being used, and have the fewest remaining tasks that require them (TDs are copied to the HD to avoid lost data).

2) Parallelizable tasks: Two approaches are studied for step 4(d) in Figure 6 of the DATCP heuristic. For the first approach, no parallelization is used. For the second approach ("max" approach), the heuristic always parallelizes tasks to multiple PEs within a compute node. Initially, the PEs that have the maximum robustness value are determined (calculated using the execution time with only one PE executing the task). We then determine which compute node has the most PEs with the maximum robustness value (multiple PEs across compute nodes can have the same robustness value), and map the task to all PEs, in that compute node, that share the maximum robustness value.

3) Satellite Data Placement: The satellite data sets are mapped by the heuristic in a simple manner. The first time an SD is required and the storage location of the SD has not been determined, the data set and the task

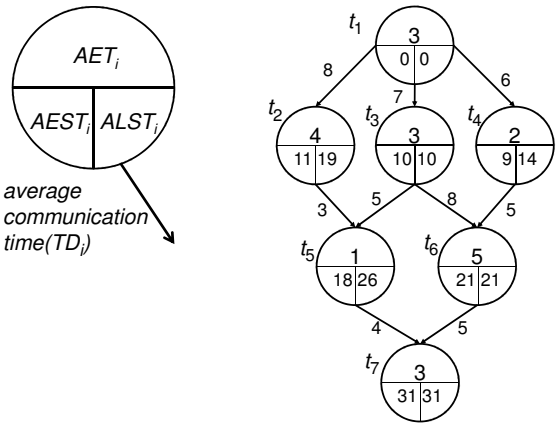

Fig. 7. Figure showing an example of AEST and ALST computation. The AEST is calculated first starting from t_1 to t_7. After all the AESTs are calculated, the ALST of t_7 is set to the AEST of t_7 (in this case 31), and the ALST of the all the tasks from t_7 to t_1 is calculated.

that requires it are mapped to the same compute node, and then the robustness value is determined. The task is assigned to the PE that maximizes robustness, and the SD is stored in the HD of this PE's corresponding compute node. Subsequent tasks requiring the data set are limited to the mapping determined by this initial task.

B. Heterogeneous Robust Duplication (HRD)

Heterogeneous Robust Duplication (**HRD**) is based on the concept of the Highest Critical Parents with Fast Duplication introduced in [10]. The algorithm has a listing phase, where tasks have their priority computed and are inserted into a queue based on that priority. The algorithm then assigns tasks to PEs in order of the queue.

This algorithm calculates two values for each task, the Average Earliest Start Time (**AEST**) and the Average Latest Start Time (**ALST**). We traverse down the DAG

(1) Traverse the DAG downward (starting at t_{entry}), computing the AEST for each task.
(2) Traverse the DAG upward (starting at t_{exit}), computing the ALST for each task.
(3) Identify all critical nodes (where $t_{entry} = t_{exit}$).
(4) Push the critical nodes on the stack ($\boldsymbol{S}$) in descending order of their ALST.
(5) While S is not empty do
  (a) If the task at the top of the stack S has a parent that is not in L then push the parent on S (so that the parent is at the top of the stack).
  (b) Else pop S and enqueue on L.

Fig. 8. Procedure used to generate the HRD list.

(1) A task (t_{map}) is de-queued from L
(2) Assign the task to the PE that results in the largest robustness value. Multiple PEs may have the same robustness value.
(3) If ties occur then we select the PE with the best MCT.
  – Multiple PEs may have the same robustness value.
(4) Repeat this process until all tasks are mapped.

Fig. 9. Procedure used to map tasks to machines for the HRD heuristic.

computing $AEST(t_i)$ for each task t_i. Let **pred(t_i)** be the set of predecessor tasks for t_i in the DAG, and **succ(t_i)** be the set of successor tasks. We can calculate $AEST(t_i)$ as follows:

$$\begin{aligned} AEST(t_{entry}) &= 0, \\ f_{pred}(t_i, t_j) &= AEST(t_j) + AET(t_j) \\ &+ ACT(dat_{i,j}), \\ AEST(t_i) &= \max_{t_j \in pred(t_i)} f_{pred}(t_i, t_j). \end{aligned}$$

We then traverse back up the DAG, computing ALST(t_i) for each task t_i.

$$\begin{aligned} AEST(t_{exit}) &= ALST(t_{exit}), \\ f_{succ}(t_i, t_j) &= ALST(t_j) - ACT(dat_{i,j}), \\ ALST(t_i) &= \min_{t_j \in succ(t_i)} f_{succ}(t_i, t_j) - AET(t_i). \end{aligned}$$

Tasks along the critical path (**critical tasks**) have ALST equal to AEST. An example of this is shown in Figure 7.

In the listing phase, a prioritized queue **L** is built for the mapping phase (procedure shown in Figure 8). In addition, we reordered the listing phase of the HRD, to queue critical tasks from the same application together, to improve the overall performance because the resultant mapping produces more efficient memory staging. The next stage of the HRD uses L (L corresponds to a total ordering of the DAG) to determine a mapping (procedure shown in Figure 9). If two possible assignments have the same robustness, then HRD uses an minimum completion time criterion to select the best solution.

After the assignment, the heuristic will duplicate the critical parent $\boldsymbol{t_{critical}}$ ($t_{critical} = \text{argmax}_{t_j \in pred(t_{critical})} f_{pred}(t_{critical}, t_j)$) if the needed conditions are met. The time slot just prior to the execution of a task is denoted as the duplicate time slot (**DTS**). The DTS for each PE on the same compute node (as the PE selected for assignment) is the start time of t_{map} minus the time when the PE finishes executing the task it has before t_{map}. If the DTS is large enough to hold the execution of $t_{critical}$ then duplication occurs; thus, eliminating the need to incur the communication cost of the $t_{critical}$'s data set. A multiplicative factor ($\boldsymbol{mult}$) is used to control the conditions required for duplication. If the DTS is larger than the result of multiplying $mult$ and the duration of the parent's execution time, then the parent can be duplicated in the DTS. Intuitively, this prevents DTS from being too close in duration to the parent's execution time, which could increase the makespan when the estimated task execution times are increased by ρ. For this study, this best value was empirically

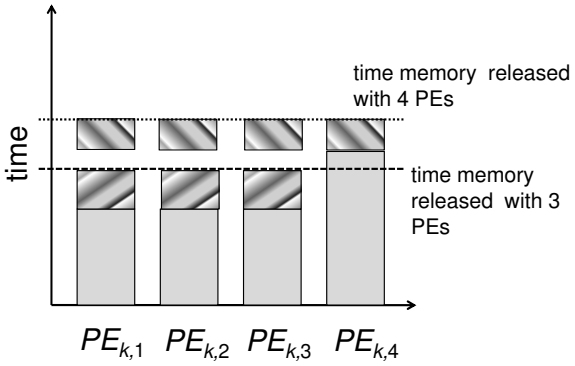

Fig. 10. Parallel example.

determined to be 1.4.

The reordering mostly eliminated the usefulness of duplicating critical parents (that was used in a different context in [10]), because the required SD and TD already reside on the same compute node as the task.

1) Memory Management: A Least Recently Used (**LRU**) algorithm is used to manage which data sets remain in RAM and which are removed. Two exceptions to a basic LRU mechanism were used. The first exception is a reference count, kept for each dataset, determined by parsing the DAG. When a data set is no longer going to be used, it is removed from memory. The second exception is for data sets currently in memory required by the next scheduled task; these data sets are not removed.

2) Parallelizable Tasks: For the HRD, if a task is parallelizable then the task parallelization that allows the compute node's memory to be released the earliest is chosen. This is illustrated in Figure 10. In this example, there are four PEs available on the compute node k, but $PE_{k,4}$ is available to start the execution of the task later than the other PEs. Parallelizing across three PEs releases the compute node's memory earlier than parallelizing across all four PEs.

3) Satellite Data Placement: Multiple scenarios are considered for satellite placement. The first is a random placement of the satellite data sets. The second is to place the satellite data set on the same compute node as the first task that requires it. Once the satellite data set is placed on a compute node, all subsequent accesses are from that compute node. The third algorithm places the satellite data sets based on a reference count. A list of SD sets is created with their associated reference counts computed from the DAG, i.e., the number of tasks that use that data set as a direct input. This list is then sorted in descending order of these reference counts. This variation will assign the same number of SD sets to all HDs. We define $\boldsymbol{SD\_per\_HD}$ as the number of SD sets per HD, i.e., $SD\_per\_HD = \frac{\#\ of\ data\ sets}{\#\ HDs}$. The first SD_per_HD satellite data sets are allocated to the HD with the largest read bandwidth, the next SD_per_HD satellite data sets are allocated to the HD with the second largest read bandwidth, and so forth.

IV. Related Work

Research about scheduling DAGs on multi-processor systems is extensive, e.g., [14], [15], [18], [21]. These studies are motivated both by manufacturing and by computational requirements. In this section, examples of existing heuristics are presented.

The Modified Critical Path (**MCP**) algorithm developed in [21] was designed for homogeneous systems and considers only one application DAG. The heuristic determines the latest possible start time of each task (constrained by the critical path length) and then creates a list of tasks in increasing order of these times. Tasks are selected for mapping in the order of the list. The selected task is then mapped to the machine that allows the earliest start time. Thus, the heuristic attempts to start critical path tasks as early as feasible. There are several differences between [21] and this study. We focus on robustness against uncertainty in execution times, and consider a heterogeneous computing environment.

The authors in [18] developed the Dynamic Level Scheduling (**DLS**) heuristic. The static level of a task is computed as an approximate time from the task node to the exit node along the worst-case path on a heterogeneous system. A data arrival time for a task is defined as the time when all required input data arrives for a task at a destination node. The dynamic level is the static level minus the data arrival time. The mappable task/machine pair with the highest dynamic level value is selected for mapping. The performance metric in [18] was makespan, and the paper did not consider uncertainties or robustness. As indicated earlier, minimizing makespan is not the same as maximizing our robustness measure for DAGs.

The Dynamic Critical Path (**DCP**) heuristic developed in [15] calculates the Absolute Earliest Start Times (**AbEST**) and Absolute Latest Start Time (**AbLST**) for each task. A task is defined to be on the critical path if its AbEST equals its AbLST. At each mapping event, mappable tasks update their AbEST and AbLST to determine which task is on the critical path. The critical path task is then mapped to a compute node that minimizes the Earliest Start Time (**EST**) of the task and the EST of its successor tasks. The DATCP heuristic differs from the DCP heuristic in that the DCP heuristic does not have to deal with uncertainty in task execution times and its performance goal is makespan rather than robustness.

The work in [17] considers a heterogeneous *ad hoc* grid used to compute an application composed of communicating sub-tasks. Both this and our study consider the mapping problem of DAGs in a heterogeneous computing environment; the heuristics in [17] minimize the average battery power consumed while meeting a makespan constraint. This paper focuses on maximizing the ability of a resource allocation to tolerate the uncertainty of execution times, while the work in [17] does not consider uncertainty. The minimization of battery power consumed, as studied in [17], is different than maximizing robustness presented in this study.

V. Results

A. Simulation Setup

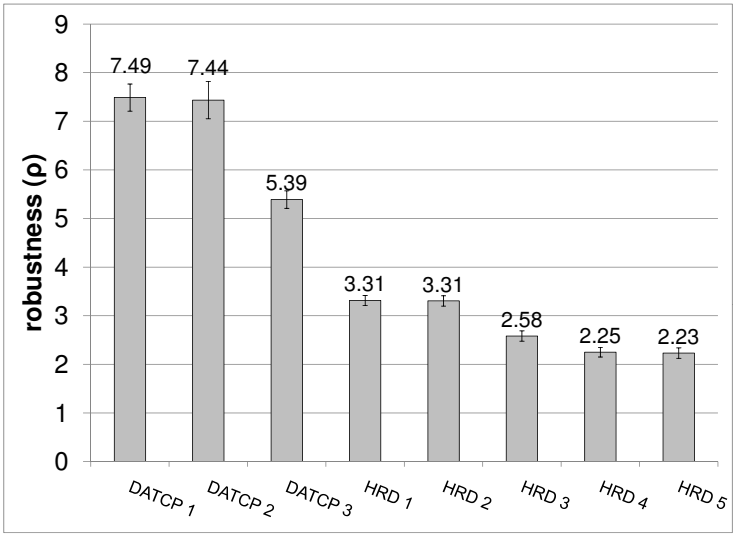

Fig. 11. Results for DATCP 1, Max parallel with satellite mapping; DATCP 2, Max parallel with random satellite mapping; DATCP 3, no parallelism with random satellite mapping; HRD 1, SD placement based on first task placement with duplication; HRD 2, SD placement based on first task placement with no duplication; HRD 3, SD placement based on reference count with no duplication; HRD 4, random SD placement with duplication; HRD 5, random SD placement and no duplication. The results are shown with a 95% confidence interval.

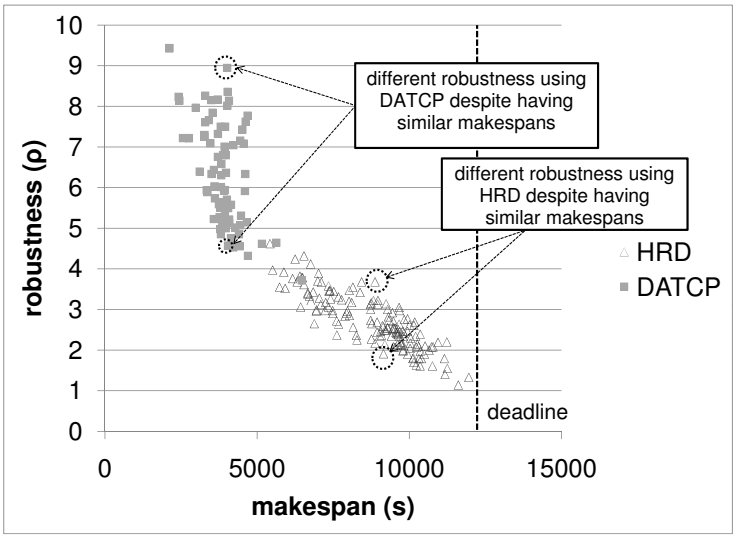

Fig. 12. Scatter plot of makespan vs. robustness for all versions the DATCP and HRD heuristics. Points in the plot represent the makespan and robustness of resource allocations generated by the heuristics described in Figure 11 (all variations). These points represent different sets of DAGs, ETCs, and communication times.

Each simulation run (50 total) has 64 unique applications, and each application is composed of 8 to 16 tasks chosen uniformly at random. For each task, the fanout range is chosen uniformly between 1 to 3, and a maximum of 3 incoming tasks and a maximum of 2 satellite data sets are allowed (number of satellite data sets is also chosen randomly). After the fanout of half the tasks in an application have been assigned, the remaining tasks are set to a fanout of 1 (to force convergence of DAG).

The size of the SD and TD sets used in this simulation varies from 1 to 20 GBytes in size. We assume the network bandwidth is 256 Mbytes per second, and that the HC system is composed of eight compute nodes.

Compute nodes 1 and 2 have eight PEs, compute nodes 3 and 4 have four PEs, and finally compute nodes 5, 6, 7 and 8 have two PEs. All compute nodes have 160 GBytes of RAM (152 are used for staging data and the remaining 8 are used to buffer data in and out of the local HD). Half of the tasks in the applications are parallelizable, with half of those having "good" parallelism and half having "poor" (see Table I). The ETC values of a task are calculated by adding all the incoming data sets (SD and TD in GBytes), and multiplying this sum by an ETC per GByte value generated using the coefficient of variation based method described in [2]. Consider that all compute nodes are sorted in descending order based on the number of PEs. If k is greater than j, then the ETC of computing t_i on compute node j is less than or equal to its ETC on compute node k. Note that even if two nodes have the same number of PEs, they may still have different ETC values. A high-task/high-compute-node heterogeneity [2] is used to simulate the tasks within the DAGs. The heterogeneity values used to generate the ETCs are $V_{task} = \frac{\sigma_{task}}{\mu_{task}}$ = 0.4 and $V_{machine} = \frac{\sigma_{machine}}{\mu_{machine}}$ = 0.3, and the mean (μ_{task}) is 1 second/Gbyte.

B. Simulation Results

As shown in Figure 11, the DATCP heuristics using "max" parallelization performed better than the DATCP variation with no parallelism. This improvement in performance can be attributed to a better utilization of PEs. Intuitively, if no additional tasks can be scheduled on a compute node due to lack of memory (i.e., the local RAM is full with data sets currently needed by the other PEs), and there are idle PEs in that compute node then the usage of parallel tasks will reduce task execution time and release the memory earlier. The improvement in performance when parallelizing tasks was on average approximately 40%.

The variations DATCP 1 and HRD 1 that placed the SD on the same node as the first task mapped to use it were the best performing variations. For this simulation, the HD speeds were faster than the network speed—making data locality the primary influence.

Comparing the results of the DATCP 1 and HRD 1 (with parallelization and with satellite data placement) to their counterparts without parallelization and using random satellite placement, we can see both have about a 40-50% improvement. The superior performance of DACTP versus HRD may be attributed to assigning the task to a PE within the node that has the most PEs that give the same robustness value. This is because PEs having equal robustness often implies that they are idle and hence permit more parallelization.

A scatter plot of makespan and robustness is shown in Figure 12. When makespan is equal to Δ then $\rho = 1$. Therefore, resource allocations with a makespan close to Δ will have a ρ near to 1; because, as the makespan gets closer to Δ there is less opportunity for ρ to increase. It is also evident that the HRD heuristic does not perform as well as the DATCP heuristic. The selected points in Figure 12 show resource allocations with similar makespans but very different robustness values. The robustness and makespan of the resource allocations in Figure 12 show why makespan is not a good measure of robustness.

VI. Conclusions

This study focused on using heuristics to do the resource allocation of multiple applications (formed by DAGs of tasks) to an HC environment based on multicore chips. The goal of resource allocations is to meet the deadline constraint while being robust against uncertainty in execution times. We modeled a heterogeneous computing system used for satellite image processing, defined a mathematical robustness metric, and created and evaluated resource allocation heuristics that maximize this robustness metric. We were able to derive a metric to measure the robustness, however, the calculation of this metric was computationally intensive.

The DATCP heuristic produced the highest average robustness values for the given simulation cases when robustness was used to guide the heuristic. The use of parallel tasks (both good and poor) improved the robustness of the solution.

The method used to break ties had a significant effect on the performance of the heuristics (selecting the compute node with the most PEs with equal robustness for the DATCP and the MCT value for the HRD). The additional information used to break ties is dependent on the fitness function used to evaluate solutions. In this study, it was possible for multiple PEs to have the same robustness. Because of this situation, a secondary fitness function (to break ties) helps improve the overall performance of a heuristic by providing additional information to differentiate between assignments with the same robustness.

Acknowledgments: *The authors thank Dalton Young for his comments.*

References

[1] S. Ali, T. D. Braun, H. J. Siegel, A. A. Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "Characterizing resource allocation heuristics for heterogeneous computing systems," in *Advances in Computers Volume 63: Parallel, Distributed, and Pervasive Computing*, 2005, pp. 91–128.

[2] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering, Special 50th Anniversary Issue*, vol. 3, no. 3, pp. 195–207, Nov. 2000.

[3] S. Ali, A. A. Maciejewski, and H. J. Siegel, "Perspectives on robust resource allocation for heterogeneous parallel systems," in *Handbook of Parallel Computing: Models, Algorithms, and Applications*, S. Rajasekaran and J. Reif, Eds. Boca Raton, FL: Chapman & Hall/CRC Press, 2008, pp. 41–1–41–30.

[4] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Measuring the robustness of a resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 7, pp. 630–641, Jul. 2004.

[5] H. Barada, S. M. Sait, and N. Baig, "Task matching and scheduling in heterogeneous systems using simulated evolution," in *10th IEEE Heterogeneous Computing Workshop (HCW '01)*, Apr. 2001, pp. 875–882.

[6] E. G. Coffman, *Computer and Job-Shop Scheduling Theory*. John Wiley and Sons, New York, NY, 1976.

[7] M. K. Dhodhi, I. Ahmad, and A. Yatama, "An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, pp. 1338–1361, Sep. 2002.

[8] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transaction on Software Engineering*, vol. SE-15, no. 11, pp. 1427–1436, Nov. 1989.

[9] A. Ghafoor and J. Yang, "A distributed heterogeneous supercomputing management system," *IEEE Computer*, vol. 26, no. 6, pp. 78–86, Jun. 1993.

[10] T. Hagras and J.Janecek, "A high performance, low complexity algorithm for compile time job scheduling in homogeneous computing environments," in *IEEE Proceedings of International Conference on Parallel Processing Workshops (ICPP03)*, Oct. 2003, pp. 149–155.

[11] 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, pp. 280–289, Apr. 1977.

[12] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42–51, Jul. 1998.

[13] A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, vol. 26, no. 6, pp. 18–27, Jun. 1993.

[14] S. Kim, S. Lee, and J. Hahm, "Push-pull: Deterministic search-based DAG scheduling for heterogeneous cluster systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 11, pp. 1489–1502, Nov. 2007.

[15] Y. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506–521, May 1996.

[16] V. Shestak, J. Smith, H. J. Siegel, and A. Maciejewski, "Stochastic robustness metric and its use for static resource allocations," *Journal of Parallel and Distributed Computing*, vol. 68, no. 8, pp. 1157–1173, Aug. 2008.

[17] S. Shivle, H. J. Siegel, A. A. Maciejewski, P. Sugavanam, T. Banka, R. Castain, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, and J. Velazco, "Static allocation of resources to communicating subtasks in a heterogeneous ad hoc grid environment," *Journal of Parallel and Distributed Computing, Special Issue on Algorithms for Wireless and Ad-hoc Networks*, vol. 66, no. 4, pp. 600–611, Apr. 2006.

[18] G. Sih and E. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175–187, Feb. 1993.

[19] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," in *5th IEEE Heterogeneous Computing Workshop (HCW 1996)*, Apr. 1996, pp. 86–97.

[20] J. Smith, V. Shestak, H. J. Siegel, S. Price, L. Teklits, and P. Sugavanam, "Robust resource allocation in a cluster based imaging system," *Parallel Computing*, vol. 35, no. 7, pp. 389–400, Jul. 2009.

[21] M. Wu and D. Gajski, "Hypertool: A programming aid for message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 330–343, Jul. 1990.

[22] D. Xu, K. Nahrstedt, and D. Wichadakul, "QoS and contention-aware multi-resource reservation," *Cluster Computing*, vol. 4, no. 2, pp. 95–107, Apr. 2001.

BIOGRAPHIES

Luis Diego Briceño is currently pursuing his Ph.D. degree in Electrical and Computer Engineering at Colorado State University. He obtained his B.S. degree in electrical and electronic engineering from the University of Costa Rica. His research interests include heterogeneous parallel and distributed computing.

Jay Smith received the B.A. in Mathematics from the University of Colorado at Boulder and the Ph.D. degree in Electrical and Computer Engineering in 2008 from Colorado State University. Jay is currently a Software Architect with DigitalGlobe and an Assistant Research Professor in the Electrical and Computer Engineering Department at Colorado State University. His research interests include heterogeneous parallel and distributed computing, robust resource management, and parallel algorithms. In the area of heterogeneous computing, he is investigating stochastic techniques for resource management, techniques for decentralized resource management, and the design and structure of large-scale distributed heterogeneous computing systems. He is a member of both the IEEE and ACM.

Howard Jay Siegel was appointed the Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU) in 2001, where he is also a Professor of Computer Science. He is the Director of the CSU Information Science and Technology Center (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. From 1976 to 2001, he was a professor at Purdue University. Prof. Siegel is a Fellow of the IEEE and a Fellow of the ACM. 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 370 technical papers. His research interests include robust computing systems, resource allocation in computing systems, heterogeneous parallel and distributed computing and communications, parallel algorithms, and parallel machine interconnection networks. He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and was 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 seven international conferences, and Chair/Co-Chair of five 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. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. For more information, please see www.engr.colostate.edu/~hj.

Anthony A. Maciejewski received the B.S., M.S., and Ph.D. degrees in Electrical Engineering in 1982, 1984, and 1987, respectively, all from The Ohio State University. From 1988 to 2001, he was a Professor of Electrical and Computer Engineering at Purdue University. In 2001, he joined Colorado State University where he is currently the Head of the Department of Electrical and Computer Engineering. He is a Fellow of IEEE. A complete vita is available at www.engr.colostate.edu/~aam.

Paul Maxwell is an active duty Army Lieutenant Colonel and a Ph.D. candidate in the Electrical and Computer Engineering department at Colorado State University. He received a B.S. degree in electrical engineering from the United States Military Academy in 1992. He received a M.S. degree in electrical engineering from the University of Kentucky in 2001. From 1992 until the present, he has served in various military positions to include platoon leader, company commander, brigade plans officer, brigade logistics officer, battalion operations officer, and battalion executive officer. He was also an Assistant Professor in the department of Electrical Engineering and Computer Science at the United States Military Academy from 2001 until 2004. During his time in service, he has deployed to the Former Yugoslavia Republic of Macedonia, the Republic of Bosnia-Herzegovina, and Iraq.

Russell Irvin Wakefield has been a Ph.D. candidate in the Computer Science department at Colorado State University since 2006. His research areas include distributed systems, security in operating systems, and access control models. He received his B.S degree in Computer Science from Colorado State University in 1980 and achieved over 25 years of industry experience in high performance computing and symmetric multi-processing systems programming at corporations including Control Data Corporation, ETA Systems, Pyramid Technology, and Evans & Sutherland. After 7 years as Director of Engineering at Cisco Systems, he returned to academia to teach and complete his Ph.D. He has been an instructor in the Computer Science department at Colorado State University since 2007. For more information, please see www.cs.colostate.edu/~waker.

Abdulla Al-Qawasmeh received his B.S. in Computer Information Systems from the Jordan University of Science and Technology in 2005. He received his M.S. in Computer Science from the University of Houston Clear Lake in 2008. Since August, 2008 he has been a Ph.D. student in Computer Engineering and a Graduate Research Assistant at Colorado State University. His research interests include robust heterogeneous computing systems and resource management in heterogeneous computing systems.

Ron C. Chiang is a Ph.D. student in the Department of Electrical and Computer Engineering at Colorado State University. He received the B.S. degree from the Department of Computer Science and Information Engineering at Tamkang University, and the M.S. degree from the Department of Computer Science and Information Engineering at National Chung Cheng University. His research interest is heterogeneous computing. He is a student member of the ACM and the IEEE Computer Society.

Jiayin Li received the B.E. and M.E. degrees from Huazhong University of Science and Technology, China, in 2002 and 2006, respectively. And now he is pursuing his Ph.D. degree in Department of Electrical and Computer Engineering, University of Kentucky. His research interests include software/hardware co-design for embedded system and high performance computing.