Ensemble Dispatching on an IBM Blue Gene/L for a Bioinformatics Knowledge Environment

Paul Marshall
Department of Computer Science, University of Colorado at Boulder
Boulder, CO 80309
paul.marshall@colorado.edu

Matthew Woitaszek
National Center for Atmospheric Research
Boulder, CO 80305
mattheww@ucar.edu

Henry M. Tufo
Department of Computer Science, University of Colorado at Boulder
Boulder, CO 80309
tufo@cs.colorado.edu

Rob Knight
Department of Chemistry and Biochemistry, University of Colorado at Boulder
Boulder, CO 80309
rob.knight@colorado.edu

Daniel McDonald
Department of Chemistry and Biochemistry, University of Colorado at Boulder
Boulder, CO 80309
daniel.mcdonald@colorado.edu

Julia Goodrich
Department of Computer Science, University of Colorado at Boulder
Boulder, CO 80309
julia.goodrich@colorado.edu

ABSTRACT

This paper discusses our work providing support for processing a large number of short tasks within the context of our development of a collaborative bioinformatics knowledge environment for structural biologists, environmental microbiologists, and evolutionary biologists. We have designed and implemented a new ensemble-based task dispatching system that we have deployed on a Blue Gene/L system in conjunction with the Blue Gene's High Throughput Computing (HTC) capability. Unlike our prior general database-backed HTC task dispatching system, the ensemble-based task dispatching system is able to efficiently process and dispatch large numbers of very short tasks to over a thousand cores. We also investigate the scalability of the IBM Blue Gene/L at HTC in general, identifying and eliminating processor-reboot inefficincies for very short tasks for specific applications, making the Blue Gene/L a feasible processing system for this bioinformatics workload.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed Systems*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Performance*

General Terms

Design, Experimentation, Measurement, Performance

MTAGS '09 November 16th, 2009, Portland, Oregon, USA

1. INTRODUCTION

In collaboration with researchers from the Department of Chemistry and Biochemistry at the University of Colorado at Boulder we are building a bioinformatics knowledge environment to support structural biologists, environmental microbiologists, and evolutionary biologists in their endeavor to understand the structures and functions of the biomolecules that are used to infer phylogeny. As emerging inexpensive gene sequencing technologies yield hundreds of thousands of sequences, the task of processing these sequences for phylogenic reconstruction is exceptionally computationally intensive and a compelling use of the high-throughput/many-task computing (MTC) facilities [14] available on current supercomputing and cluster computing systems.

From a workflow perspective, the applications we use for phylogenic reconstruction are quite similar and well-suited for MTC. We have deployed three of these applications to date: Clearcut [7], FastTree [13], and RAxML [16]. All of these programs accept arguments on the command-line controlling simulation parameters, read short input data files, and produce short output text files. For the sequences we are evaluating, the execution times fall into two categories: ClearCut and FastTree have relatively short running times (from tens of seconds to tens of minutes), while RAxML typically runs for tens of minutes up to a few hours. In a typical experiment, many thousands of sequence input files are evaluated by separate executions of the software, occasionally using several command-line argument variants to examine the effect of parameters. Thus, the short execution time and the need to process many of individual program executions presents a workload typical of MTC.

In 2007, when IBM released support for HTC on the Blue Gene/L series of systems, we developed a general-purpose software solution to allow our user community to utilize the HTC feature [5, 12]. Prior to the introduction of HTC support, the Blue Gene/L was capable of allocating processors only in 64-processor partitions intended for MPI parallel programs. IBM's support for HTC allowed individual

processors in a partition to run different executables (albeit under the auspices of a single user's controlling queue job), and using the HTC features to run an application required custom software development to manage the tasks. In our implementation, tasks were stored as command lines in a SQL database and task management software submitted jobs to allocate partitions via the system scheduler on the user's behalf, dispatched tasks to the appropriate partitions when the corresponding jobs started running, and updated the database with task return codes. This system worked quite well for the 30-60 minute tasks required by the terrestrial carbon cycle model that provided the motivation to implement HTC support on our 2048-core Blue Gene/L at the time.

Unfortunately, our original general-purpose HTC solution collapsed under the increased task dispatch rate generated by the much shorter bioinformatics tasks. Through iterative analysis, we identified two bottlenecks to decreasing time scalability: the overhead of the database-based task dispatch, and the reboot time of the Blue Gene/L nodes themselves. We reduced the task dispatching bottleneck by adopting an ensemble-based task specification strategy that delays the enumeration of tasks until execution, eliminating the need to store tasks and their status in a persistent database. This dispatching method still supports the execution of generic single-thread executables on the Blue Gene/L using HTC.

In cases where the Blue Gene/L node reboot time was itself problematic, we eliminated the node reboot time by the rather drastic action of moving the task construction code from system software into the target executable itself. That is, instead of running generic unmodified executables with inefficient system processes, we integrate native support for HTC-style task execution directly into the target executable, removing the requirement to reboot the processors between each task. Combined with the prior ensemble-based workflow management, this solution supports running the most critical applications with minimal HTC overhead at the cost of software development time.

The remainder of this paper is organized as follows: Section 2 presents our design, describing the ensemble-based task specification method and the mechanism to eliminate Blue Gene/L node reboot time when necessary. Section 3 describes our implementation of this architecture, and is followed by performance measurement results and a discussion that considers this solution within the context of related work. The paper concludes with the performance results of a representative bioinformatics run and future work.

2. DESIGN

Our HTC task dispatching and launching system is designed to support the IBM Blue Gene/L as its primary computational platform, but the design is not limited to this particular resource. The overall design utilizes a worker-initiated task assignment methodology. Unlike batch schedulers that push computational jobs to idle nodes, an idle resource requests work by executing a task launcher program. The task launcher contacts a central service that selects and dispatches a task from its pool of pending work. In general, tasks are dispatched as command lines consisting of the executable name, arguments, and additional environment variables, allowing any system capable of requesting work from the centralized dispatcher to receive tasks for processing.

Within this framework, we present three design variants that differ in their approach to generalization for both task specification and application software. In the first case, a generalized task dispatch mechanism with a generalized HTC task launcher provide support for executing arbitrary executables, as is typical of a general-purpose HTC or MTC solution. In the second case, we encapsulate (but do not necessarialy constrain) task specification into ensembles while still relying on the generalized task launcher. For cases where ensembles execute well-known workflows and small sets of applications, we optimize the client application to support HTC task dispatching directly, eliminating the overhead of the generalized task launcher. Each of these design variants is described in detail below.

2.1 General Dispatching with General HTC Launching

The first design variant uses a general task dispatcher with a general HTC task launcher to provide complete flexibility in running arbitrary single-processor executables on the IBM Blue Gene/L in HTC mode (see Figure 1). A batch job to run a partition in HTC mode is submitted to the system's Cobalt scheduler. The scheduler allocates a partition to the user, and then the Blue Gene/L HTC control system starts our task launcher on every node/processor in the partition. The task launcher opens a TCP connection to a centralized dispatcher that retrieves a task from a database-based work pool and transmits it to the launcher; the launcher then starts the task using `execve`. When the task executable terminates, the Blue Gene/L control system reboots the processor and invokes the task launcher, and the process repeats with the next task.

In our prior work [5], we used a PostgreSQL database to store task definitions, wrote Globus GRAM and WS-GRAM [8] front-ends to serve as a familiar user interface for "real" users (while we continued to directly place work in the database using SQL), and dispatched tasks directly from the database. As the number of tasks increased, dispatching tasks directly from the database became increasingly time-consuming, and the addition of a prefetching mechanism to the dispatcher provided a temporary workaround allowing the system to continue to function in production.

The large number of short tasks presented by the bioinformatics use case completely overwhelmed this task dispatching system. The mere management activities of loading sets consisting of many thousands of new tasks into the database and reading the results from previous ensembles, while the dispatching system was locking task records for dispatch and result disposition recording, reduced even dedicated database servers to a crawl. As a solution, we developed the ensemble task dispatching system described next.

2.2 Ensemble Dispatching with General HTC Launching

The second design variant uses an ensemble-based workflow management paradigm. Instead of keeping track of every individual task as a discrete item in a memory structure or a database, entire data sets and the operations to be performed upon them are encapsulated in a simulation ensemble. In our implementation (described in detail later), an ensemble is based on a Python generator function that returns an iterator that can be used to produce command lines representing tasks on demand. For example, for the

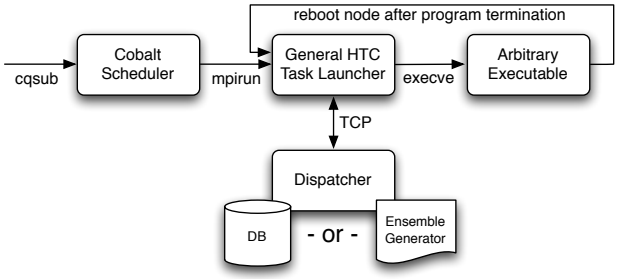


Figure 1: General and ensemble task dispatching on an IBM Blue Gene/L with a generalized HTC task launcher.

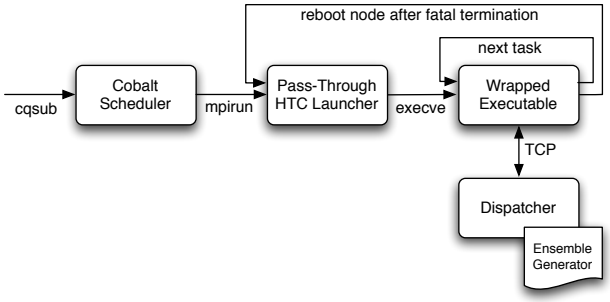


Figure 2: Ensemble task dispatching on an IBM Blue Gene/L with the HTC task launching code wrapping the target executable.

bioinformatics applications, the generator invokes the target application several times for each parameter variant for every input data file that was provided by the biologist.

The operation of the ensemble dispatching mechanism is identical to the task dispatch process described previously (see Figure 1), but the tasks are dynamically generated by the ensemble instead of retrieved from a database. The ensemble concept also maps nicely to many typical use cases of MTC, such as parameter studies and running programs over each file in a data collection. The ensemble neatly encapsulates common configuration information for an experiment, such as the names of the executables and the parameters to be selected, while allowing run-time expansion against dynamically specified data sets. The entire ensemble may then be submitted through a typical workflow paradigm, with the last stage expanding the ensemble definition into the many hundreds of thousands of individual executions. By delaying the ensemble expansion to the final computational resource, the burden of tracking remote progress through the entire system (e.g, on a Grid or Cloud) is greatly reduced. The system-level resource managers need to manage only a single ensemble job, not the many individual tasks.

2.3 Ensemble Dispatching with HTC-Enabled Task Launching

The ensemble-based dispatching reduces the overhead of assigning tasks to workers, but for exceptionally short tasks the processor reboot time of the IBM Blue Gene/L becomes a critical bottleneck. We explored the Blue Gene/L's HTC capabilities by creating a specialized task launcher that executed a hardcoded program – a 'null task' that records its timestamp and exits. By counting the number of executions over a time window, we determined the amount of time required by the Blue Gene/L to boot a node in HTC mode and execute the launcher. The Blue Gene/L HTC reboot-launcher-exec paradigm indeed does not scale well for very short tasks. If an entire 64-core partition is configured to run null tasks, the task cycle time is approximately 96 s/task. That is, after a task completes, 96 s will elapse until the processor is available to request its next assignment. This corresponds to a task dispatch rate of .66 tasks/s for a 64-core partition; a severe limitation also noticed by Raicu et. al. when developing Falkon [15].

However, it is important to note that this measured 96 s/task cycle time is not a constant feature/limitation of the partition, but is influenced by the task workload on the partition as related to the length of execution of the tasks. If only 1 out of the 64 cores is assigned to run tasks, the Blue Gene can achieve a task cycle time of 1.65 s/task; similarly, if the tasks require minutes instead of seconds to execute, the task cycle time overhead is substantially less. (Full results are presented in Figures 3, 4(a), and 4(b) and discussed in the Results section.) For exceptionally short tasks, though, the IBM Blue Gene/L reboot-launcher-exec paradigm is prohibitively inefficient.

The third design variant optimizes the execution of programs with very short run times by moving HTC task assignment support directly into the target executable itself (see 2). When booted by the Blue Gene/L control system, a simple pass-through launcher immediately executes the HTC-enabled application executable specified by the command line. The HTC-enabled executable consists of the original program plus wrapper code that replaces the program's main() routine. The wrapper contacts the dispatcher, retrieves a task assignment suitable for the program, and then calls the application's main() routine to perform the processing. If the wrapped application terminates successfully, the wrapper immediately requests another task, so no node reboot and launcher invocation are necessary. Should the application fail, the Blue Gene/L control system reboots the pass-through launcher. This restarts the application, which then continues to request and process the next test. Thus, in the best case, the reboot overhead is completely eliminated. In the worst case, where programs crash or abruptly exit with error codes, the performance degrades to the prior generalized-launcher case.

The switch from general HTC task support to customized HTC-enabled applications clearly eliminates the generality of the dispatching system. However, with an appropriately designed dispatcher, all three design variants can exist simultaneously using the same software framework. Because dispatchers and launchers do not require a one-to-one mapping, and dispatchers can produce tasks from databases and generators, arbitrarialy complex workflows can be supported by combining these components as required. For example, high-profile applications such as the three used for our bioinformatics research can be wrapped with the HTC code to greatly reduce their execution overhead, while other unmodified applications can be executed using the generalized components. The first design fits arbitrary use cases, ensemble

dispatchers are useful for workflows with many similar executions, and HTC-enabled applications are ideal for regularly executed computationally-intensive production workloads with very short tasks.

3. IMPLEMENTATION

We implemented our ensemble-based dispatching architecture with general task launching as well as the task-specific version of HTC. Both versions utilize the same ensemble dispatcher. The ensemble dispatcher is written in Python and uses TCP to communicate with the HTC launcher running on the compute nodes using a custom protocol.

Ensembles are defined by small, custom Python modules that contain functions that implement generators. For the bioinformatics case, each generator is configured with lists specifying the programs to be run, the required parameters for each invocation of each of the programs, and references to the data directories containing the input data and the location for the output. The generator is thus capable of returning a single next task every time it is called by the dispatcher.

In order to provide rudimentary fault tolerance and task-level control, the generators for the bioinformatics applications establish state by looking for output files on disk. As the generator is iterated forward, tasks with proper output files are considered complete and skipped. Thus, in the event of a total system crash, the generator can be replayed from the beginning, and only tasks that did not complete successfully are dispatched. Of course, this requires verifying the existence of, and possibly reading, large quantities of small output files from a file system – a potentially time-consuming process for large task sets with many files. However, because the output files are produced in the general course of execution (they are always there anyway) and are required for the bioinformatics postprocessing to establish the results of the experiment, relying on them in this nature solves the task completion detection problem using a pre-existing artifact of execution.

Experiments are managed by pairing dispatchers with ensemble generators. When multiple ensembles need to be processed we create a generator for each of the ensembles containing the required programs and their parameters as well as pointers to the input and output directories for the ensemble.

The task-specific design replaces the generic launcher program with wrapped executables that contain code to communicate with the dispatcher to request and receive task assignments. This requires modifying the source of the program that we wish to run on the compute nodes. While this could probably be achieved by linker tricks, doing so is potentially hazardous and done only occasionally – so we currently perform this step manually.

In order to make a program natively HTC-enabled, the original main() function must be overridden and replaced with one that calls the original main() function many times. To do this, we rename the main function in the original program, ensure that the program returns on successful completion (as opposed to explicitly calling exit), check that the program contains no obvious bad memory management practices, and verify that all global variables are reinitialized properly. This last step is particularly important as many programmers are tempted to allocate memory, use it throughout the program, and leave it until the program terminates. Since the application's main() routine may now be called many times, proper program memory management is essential to success, in terms of both executing without crashing and producing correct results. Once the program has been wrapped and validated, it can request tasks from an appropriately configured dispatcher.

4. RESULTS

In order to gain an understanding of the capabilities of our IBM Blue Gene/L system at HTC, we first evaluated the system using hardcoded test cases designed to measure the efficiency of the system absent any custom task dispatching software. We then examined the standard HTC use case, using general dispatchers and a general task launcher, and measure the overhead introduced by the dispatching system. Finally, we moved the HTC support code into the application, and measured the performance without the Blue Gene/L node reboots. This case also more fully exposes the scalability and limitations of the ensemble-based task dispatching system.

As a metric, we define *task cycle time* to be the elapsed time between executions of tasks of varying known run times. For our analysis we consider task run times of 0, 10, 30, 60, 120, and 300 seconds. The first case, which reports the elapsed time between the executions of tasks that sleep for zero seconds, is referred to as the *null-task cycle time*. The null-task cycle time identifies the overhead required to execute a serial task in HTC-mode on a Blue Gene/L under pathological scalability conditions, as the system merely processes tasks without performing any useful work. For the other test cases, the tasks sleep a known predefined amount of time. (In our implementation, the tasks sleep to within 1 second of the configured time, but never exceed the specified sleep time.) The reported actual task cycle time is the elapsed time for an individual task in its entirety including overhead. In addition to the task execution time, it includes overhead such as the amount of time required for the system to boot the node and start the task launcher, and for the task launcher to communicate with the dispatcher and launch the task.

4.1 Blue Gene/L HTC Capability Analysis

We first examined the amount of time to execute null tasks on Blue Gene/L partitions, first ranging from 1 to 64 processors on a single partition, and then for multiple 64-processor partitions from 1 to 16 partitions (see Figure 3, "Blue Gene/L capability test" line). For undersubscribed partitions, the task cycle time increases linearly with the number of dual-core nodes in use, ranging from 1.5 s/task to 96-100 s/task. Once a partition is fully subscribed, the task cycle time is constant at ~96 s/task. That is, regardless of how many 64-processor partitions are added, each fully-subscribed partition maintains its ~96 s/task cycle rate. In our prior work [5], we observed that dispatching tasks with executables of 1 MB, 2 MB, and 8 MB in size did not appear to be directly related to this task cycle overhead.

The fully subscribed partition task cycle rate is independent of the Blue Gene/L processor mode selection. The null task timing results – including the ~96s/task cycle rate for a 64-processor partition – are similar regardless if the partition is executing 32 tasks on a 64-processor partition in coprocessor mode or 64 tasks on the same partition in virtual node mode. We suspect that this bottleneck is related

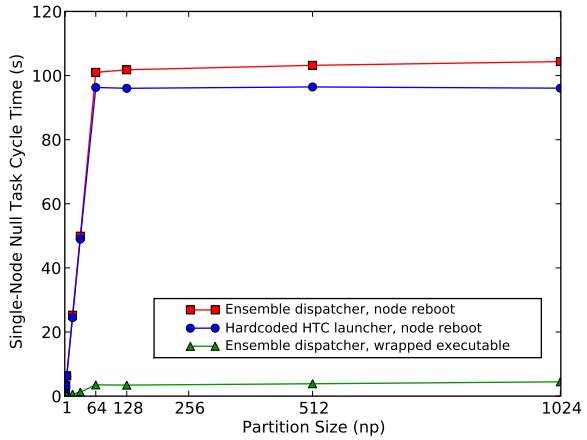


Figure 3: Null-task cycle time by partition size on an IBM Blue Gene/L. The "hardcoded HTC launcher, node reboot" test case measures the node reboot time of a static program without using any custom task dispatching features, thus reporting the performance of the Blue Gene/L rack itself.

to the mechanism used by the Blue Gene/L to boot processors in HTC partitions. When a processor boots, its kernel must be transferred to the node over the 100 Mbps JTAGS service network. Each partition contains one JTAG connection, where nodes utilize it in a serial fashion creating contention when 32 nodes attempt to boot. The task cycle time is constant with the addition of partitions, demonstrating that multiple partitions are able to use the JTAG network in parallel for a system of our size.

Because undersubscribing Blue Gene/L partitions is unlikely in practice, especially for MTC applications, the ˜96 s/task cycle time is the worst possible overhead that may be encountered by applications executing tasks of short duration. Of course, not all MTC applications execute extremely short tasks, so to expore the system's behavior further we tested applications with an increasing amount of computational processing time.

As task run time increases, the boot overhead is consumed by the task execution time (see Figure 4, "Blue Gene/L capability test" line). When task executions are 120 seconds or longer, the boot overhead of individual nodes is reduced to a few seconds. Even for large numbers of processors, the task cycle time for 300 second tasks is within a few percent: 318 s for np=1-24, and 326 s for np=2048. As task length increases, the frequency of reboots decreases, which reduces the condition that causes the contention, thus reducing the amount of time required to reboot the node and load the next task.

4.2 Task Dispatching and Launching Results

The addition of the general purpose HTC task dispatching and launching infrastructure adds a small overhead to the baseline Blue Gene/L processor cycle time. For the null-task test, which highlights the overhead using pathologically small tasks, the dispatching and launching adds 4 to 8 seconds to the task cycle time (see Figure 3, "ensemble dispatcher, unmodified executable, reboot" line). For example, for np=64, the cycle time increases from 96 to 101 s/task; for np=1024, the cycle time increases from 96 to 104 s/task.

For fixed partition sizes, the overhead of the dispatch and launch process is expected to remain constant regardless of task run time. This is the case for the 64-processor case (see Figure 4(a)), but not for the 1024-processor test case (see Figure 4(b)). In the latter case, the overhead of the dispatch and launch process also appears to increase as the task run time increases For example, for np=1024 with 300 second tasks, the Blue Gene/L task cycle time was 318 s/-task, but with the dispatcher, the task cycle time increases to 405 s/task. It is tempting to attribute this additional overhead to the dispatcher itself, but this is clearly not the case: our tests using the ensemble dispatcher but not the general-purpose task launcher indicate a task cycle time of 305 s/task for 300-second tasks. Thus, for these 300 second tasks, about 5 seconds/task is the dispatch overhead, while the remaining ˜95 s/task overhead must be attributed to the general-purpose task launcher. We intend to run further test cases to confirm this result with intermediate and larger partition sizes and task run times.

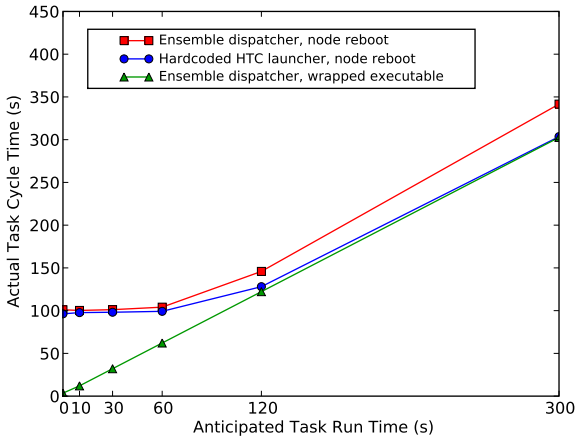
The most efficient method to dispatch and execute tasks is to create an HTC-enabled executable and avoid the Blue Gene/L reboot-launcher-exec overhead. Because the HTC-enabled executable doesn't require a node reboot for each task, the only overhead incurred (beyond the initial boot) is the function call and network communication to request the next task from the dispatcher (see Figures 3 and 4, "ensemble dispatcher, wrapped executable, no reboot" line).

In addition, the ensemble dispatcher also appears to scale well in our tested range of 64-2048 processors, even for exceptionally small task turnaround times. The null task test shows a task cycle time of ˜3-5 s/task for np between 64 and 1024, and ˜8 s/task for np=2048. This is substantially less than the ˜96 s/task cycle time encountered using the reboot-launcher-exec procedure. For increasing run times, the run time eclipses the overhead, resulting in total task run times within single-digit percentages of the application run time.

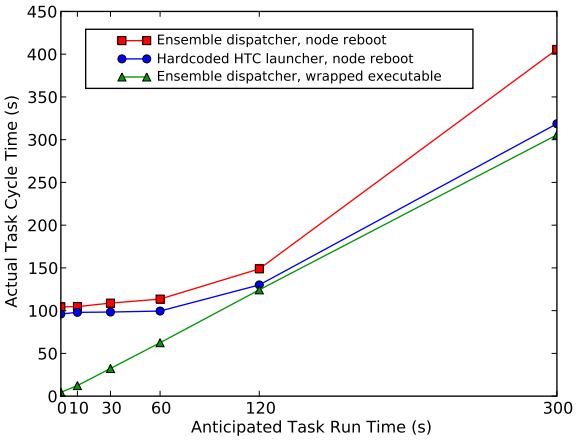
5. BIOINFORMATICS TEST CASE

The short running time of FastTree and Clearcut make them ideal candidates to be enhanced to support HTC natively. We created an HTC-enabled version of Clearcut to compare against a generic version of Clearcut; the generic version used the generic HTC launcher. Creating the HTC-enabled version of Clearcut involved three simple changes: changing the definition of main in clearcut.c to clearcut_main, having the function return at the end of execution instead of exit, and finally, changing the main function in our wrapper to call clearcut_main. Implementing these changes and recompiling Clearcut for the Blue Gene compute nodes only required minutes of software development time, although validating their results required additional time. We also created a generator specifically for our Clearcut test case (see Listing 1 for example pseudocode). The generator specified the path to the executables as well as the arguments used.

For the test procedure, we configured two ensemble dispatchers, both with the same set of input files and parameters. The test used a single parameter combination, 12,600 unique input files, and 256 compute nodes with 512 cores. Switching from general task launching to using the HTC-



(a) 64-core partition



(b) 1024-core partition

Figure 4: Actual task cycle time vs. anticipated task run time (overhead) for HTC variants on a 64-core and 1024-core IBM Blue Gene/L partitions. The "hardcoded HTC launcher, node reboot" test case measures the node reboot time of a static program without using any custom task dispatching features, thus reporting the performance of the Blue Gene/L rack itself.

enabled version reduced the total execution time from 2694 seconds to 844 seconds, eliminating 1850 seconds of overhead from the execution time, resulting in a speedup of 3.19.

As a result of our successful test case, we have also created an HTC-enabled version of FastTree and are in the process of validating its correctness. In the future, we plan to do the same for the other applications that can make use of MTC on our Blue Gene/L within the context of our bioinformatics knowledge environment.

6. DISCUSSION AND RELATED WORK

The efficiency of dispatching short tasks to many computational nodes has also been highlighted in the related work of several other groups in the literature. The overhead introduced by the Blue Gene/L's process of rebooting a processor before running every individual task was identified as a limitation by Raicu et al. in their development of the Falkon MTC lightweight task execution framework [15]. Because of this limitation, the authors concluded that the Blue Gene/L was insufficient as a platform for supporting a massive amount of short-running tasks and prefer the Blue Gene/P due to its support of the `fork` system call. We confirm that there is significant overhead when using Blue Gene/L HTC as a generalized task execution environment for tasks under 120 seconds, but find that this overhead is absorbed if the tasks are longer than 120 seconds. Furthermore, the reboot overhead may be completely eliminated if the individual programs are wrapped with the necessary code to implement the HTC functionality without the general-purpose dispatching.

Raicu et al. also present their architecture, Falkon, for managing the execution of a large number of tasks. Falkon is similar to our prior work in that it is a general task dispatcher capable of achieving much higher task throughput than traditional resource managers alone, including, Cobalt [6], Condor [17], or PBS [3]. We do not provide a comparison between Falkon and our lightweight Python ensemble dispatcher.

From a workflow management perspective, the ensemble dispatching method has obvious benefits in terms of reducing the complexity of the controlling workflow. When executed from a Grid framework, not only must tasks be run, but files staged in and out from the workflow management system. Just as tracking many tasks independently is difficult, staging an equal (or greater) number of input and output files presents a similar management burden. For example, on our Blue Gene/L system originally configured for high-performance computing applications with few very large files, the bioinformatics applications began to strain the inode availability of our large-scale high performance file systems if the experiments were not carefully managed. Ensemble-based management ensures that experiments are ingested, executed, and packaged and returned to the workflow manager as a unit, which greatly simplifies the workflow management execution and file staging operations.

For general tasks, Hui et al. [11] propose a lightweight execution framework called Gracie intended to provide efficient task management on heterogeneous Grid resources. Gracie is built using web services and groups tasks together into a single request that are sent to distributed resources for execution. Gracie differs from our task dispatching architecture by its focus on Grid resources as opposed to our focus on the Blue Gene architecture. Our solution also differs from Gracie at the point of individual task specification. In Gracie tasks are generated and stored in single pool, which is then divided into smaller groups and dispatched to distributed Grid resources. In our solution individual tasks are not enumerated, they are generated dynamically at the time of execution.

One obvious drawback of our choice to move HTC task support from a general-purpose task launcher and into an application itself is the loss of generality and introduction of software development work to perform the transition. (This is separate from our use of ensemble-based task dispatchers,

Listing 1: Bioinformatics ensemble generator pseudocode

```
def generator(EnsembleName, ProgramName, BaseDirectory):
    InputDirectory = BaseDirectory + EnsembleName + /input/
    OutputDirectory = BaseDirectory + EnsembleName + /output/

    if ProgramName == Clearcut_Original:
        ClearcutExe = Full Path to Original Clearcut Executable
    else if ProgramName == Clearcut_HTC:
        # Executable path validated by running HTC-enabled program
        ClearcutExe = Full Path to HTC-enabled Clearcut Executable

    # Specify, as a list, argument combinations that should all be executed
    # against the same input file
    ArgumentsToSweepOver = List of Specific Clearcut Arguments

    for InputFile in InputDirectory:
        for ClearcutArguments in ArgumentsToSweepOver:
            # Create a unique file name, the file name is typically a combination of
            # the ensemble name, program name, input file name and arguments used
            OutFileName = EnsembleName + ProgramName + InputFile + ClearcutArguments
            StdOutFile = OutputDirectory + OutFileName.stdout
            StdErrFile = OutputDirectory + OutFileName.stderr

            if StdOutFile Exists and StdErrFile Exists:
                pass
            else:
                yield (ClearcutExe, ClearcutArguments, InputFile, StdOutFile, StdErrFile)
```

which may be used in either case.) The customization of executables to support HTC natively is only required when the target workflow must execute many extremely small tasks, and only when that workflow needs to be executed on a Blue Gene/L. It could be characterized as a last-ditch effort to make the execution of those workflows *possible* on the Blue Gene/L, but brings up the broader issue of software specialization. If a program is to be run for millions of executions on a particular architecture, consuming multi-million dollar machines for processor years, we believe it is appropriate to consider discarding a fully general solution in favor of one that minimizes execution overhead, balancing investment in infrastructure and software development.

There are certainly other architectures and frameworks capable of efficiently processing this bioinformatics workload, including generic Linux clusters or Cloud-based infrastructures running software frameworks such as Hadoop [4]. However, our primary large-scale computing system is a 8192-processor Blue Gene/L, which is our main reason for focusing on its ability to support this task execution profile.

7. FUTURE WORK

Our current deployment focuses on the Blue Gene, which is an example of a tightly coupled deployment of the underlying applications with a specific resource. An emerging trend in industry [1] and the scientific computing community [10] is a move toward cloud-based virtual clusters. In particular, there is research focusing on "impromptu clusters," or clusters capable of spawning a large number of tasks encased in virtual machines to idle nodes, which begin executing in under a second [2]. Such a cluster would match perfectly with the embarrassingly parallel and short-lived nature of Clearcut and FastTree tasks.

Our current work is only a preliminary step toward the larger bioinformatics knowledge environment. Our ensemble dispatcher will need to be integrated with the knowledge environment and distributed across the Grid [9]. The generators will need to be created and managed dynamically. A key emphasis of the knowledge environment will be data annotation and provenance capabilities. Data annotation and provenanace will impact all levels of the workflow process, including the tracking of input and output data for all ensembles. Ensembles will need to be automatically deployed and archived at appropriate times to ensure timely processing of the data as well as minimizing the impact to the storage resource.

8. CONCLUSIONS

In this paper we explored the MTC characteristics of an initial set of applications that will comprise our bioinformatics knowledge environment. In particular, the knowledge environment will need to support millions of short running tasks. Thus we have also explored, in depth, the feasibility of the Blue Gene/L as a computational platform for this bioinformatics execution profile. We conclude that the Blue Gene/L is capable of efficiently managing and executing a large number of short running tasks. While longer running tasks do not require modification, tasks under 120 seconds should be customized to enable subsequent executions without forcing a reboot on the node.

In addition, we developed a generic lightweight ensemble-

based task dispatcher that maintains minimal state in RAM and is capable of quickly resuming execution by examining the input and output directories of a specific ensemble. Our ensemble dispatcher integrates seamlessly with the generic HTC launcher as well as the task-specific HTC-enabled executables.

9. ACKNOWLEDGMENTS

We would like to thank Theron Voran for his assistance with effectively using our Blue Gene/L's HTC support and Michael Oberg for his work on the original C-based HTC launcher code used as the basis for this specialized implementation. The original C launcher code is based on the IBM example reference implementation for HTC on the Blue Gene/L.

10. ADDITIONAL AUTHORS

Additional authors: Jeremy Widmann (Department of Chemistry and Biochemistry, University of Colorado at Boulder, email: `jeremy.widmann@colorado.edu`).

11. REFERENCES

[1] Amazon Web Services. http://www.amazon.com/aws/.

[2] H. Andres Lagar-Cavilla, J. Whitney, A. Scannell, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Impromptu Clusters for Near-Interactive Cloud-Based Services. *Department of Computer Science, University of Toronto, Technical Report*, June 2008.

[3] B. Bode, D. Halstead, R. Kendall, Z. Lei, W. Hall, and D. Jackson. The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters. *Usenix, 4th Annual Linux Showcase and Conference*, 2000.

[4] D. Borthakur. The Hadoop Distributed File System: Architecture and Design. *Hadoop Project Website*, 2007.

[5] J. Cope, M. Oberg, H. Tufo, T. Voran, and M. Woitaszek. High Throughput Grid Computing with an IBM Blue Gene/L. In *IEEE International Conference on Cluster Computing*, September 2007.

[6] N. Desai. Cobalt: An Open Source Platform for HPC System Software Research. *Edinburgh BG/L System Software Workshop*, 2005.

[7] J. Evans, L. Sheneman, and J. Foster. Relaxed Neighbor-Joining: A Fast Distance-Based Phylogenetic Tree Construction Method. *Journal of Molecular Evolution*, 62:785–792, 2006.

[8] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *IFIP International Conference on Network and Parallel Computing*, pages 2–13, 2005.

[9] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15:200–222, 2001.

[10] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the Use of Cloud Computing for Scientific Workflows. In *SWBES*, December 2008.

[11] L. Hui, Y. Huashan, and L. Xiaoming. A Lightweight Execution Framework for Massive Independent Tasks. In *Many-Task Computing on Grids and Supercomputers*, November 2008.

[12] A. Peters, A. King, T. Budnik, P. McCarthy, P. Michaud, M. Mundy, J. Sexton, and G. Stewart. Asynchronous Task Dispatch for High Throughput Computing for the eServer IBM Blue Gene Supercomputer. In *IEEE International Symposium on Parallel and Distributed Processing*, 2008.

[13] M. Price. FastTree. http://www.microbesonline.org/fasttree/.

[14] I. Raicu and I. Foster. Many-Task Computing for Grids and Supercomputers. *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08)*, 2008.

[15] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, and B. Clifford. Toward Loosely Coupled Programming on Petascale Systems. *Proceedeings of the 2008 ACM/IEEE conference on Supercomputing*, November 2008.

[16] A. Stamatakis, T. Ludwig, and H. Meier. RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21:456–463, 2005.

[17] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.