BigJob Experimentation

(1) *RADICAL, Rutgers University, Piscataway, NJ 08854, USA*
(*) *Contact Author:* shantenu.jha@rutgers.edu

***Abstract*—**

I. Introduction

Pilot-Job systems **??** provide a decoupling of workload submission from resource assignment. The benefits of Pilot-Jobs include allowing new execution models, simplifying distributed workload management, and unifying distributed heterogeneous computational resources. However, by providing an interfacing layer between resources and computational tasks, Pilot-Jobs also present some additional overhead. To most effectively deploy Pilot-Jobs across large-scale DCI, an understanding of Pilot-Job performance characteristics must be reached.

BigJob, a Pilot-Job implementation, was benchmarked in a variety of experimental configurations to gain an understanding of Pilot-Job performance characteristics. BigJob's operations as they relate to Pilot-Jobs were decomposed, and a model was constructed to make predictions on Pilot-Job performance using BigJob as a base.

- What are the general scaling characteristics of Pilot-Jobs?
- Where would our work best be directed to improve Pilot-Job scaling characteristics?
- What is the theoretical limit of the throughput (number of jobs) that Bigjob can support as function of number of agents? size? or any other parameters? (HTC: A measure of number of tasks as well as a number of tasks per unit time)

II. Benchmark Architecture

To measure the performance of BigJob, a two-pronged approach was taken. BigJob's external performance characteristics were collected by executing a variety of BigJob experiments in various configuations and gauging externally-measurable properties such as time-to-completion, CU submission time, and so on. Internal performance characteristics were captured by modifying the BigJob agent to collect profiling information.

A. External Benchmarking

The external benchmarking information is collected via a Python script which invokes BigJob in the usual manner for a BigJob application. In other words, no internal/private BigJob functionality is explicitly accessed in order to capture "real-world" performance characteristics. The benchmark's execution model is similarly constructed to behave as a usual BigJob application might run. For each experimental configuration, pilots and CUs are created and CUs are executed as quickly as possible without any CU-level dependencies. Benchmark timing begins once all pilots are online, at which point CU execution begins. CU runtime information is collected directly from CU accessor methods, where other information such as BJ TTC are calculated from timers within the benchmarking script itself.

B. Internal Benchmarking (Instrumented BJ)

Discussion of Pilot-Agent instrumentation.

III. Experiment Description

The experimental scenarios were designed to operate on production hardware at-scale under identical conditions as BigJob-enabled scientific software.

A. Experimental Environment

The experiments were run on HPC environments on XSEDE/TACC's Stampede supercomputing cluster shared simultaneously with hundreds of other users and thousands of other jobs. This presents fluctuation which varies due to system load and other factors, as opposed to a simulated grid environment.

B. Experimental Design

BigJob's execution model was first examined to determine the most important characteristics of BigJob. The primary inputs instrinsic to BigJob were the number of Pilots, the number of cores per Pilot, and the composition of CUs (number of CUs, CU task length, number of cores required). One important input was found to be the key-value pair server used and its effect on latency to the resource executing the BigJob benchmarking script.

To characterize BigJob's scaling performance, the total time to completion (TTC), number of CUs executed per second, CU queuing time, CU submission time, and the per-CU execution time were collected.

1) Strong scaling experimentation, local REDIS: This experimentation operates with a fixed workload of CUs, a single HPC resource, and a REDIS server co-located on the HPC resource. The number of pilots, the size of each pilot, and the number of threads may vary.

2) Weak scaling experimentation, local REDIS: This experimentation operates with a workload of CUs proportional to the number of BigJob processes, a single HPC resource, and a REDIS server co-located on the HPC resource. The number of pilots, the size of each pilot, and the number of threads may vary.

3) Remote REDIS experimentation: This scenario simulates a complex multi-resource application load. Increased ratio of pilots:workload, multiple resources (e.g. Stampede, Lonestar, Trestles), non-local REDIS.

IV. Experimental Results

The information collected from the runs consists of over fifty columns of data.

A. Data and Figures

V. Analysis

A. REDIS

The latency of the REDIS service has a major impact on performance. Figure 3 illustrates the impact of REDIS performance on a single pilot.

B. Compute-Data Services

Inefficiencies were found in an earlier version of Compute-Data and patched. Figure**??** shows this. Preliminary results (not presented) show improved performance.

C. Per-CU Behavior

Figure2 shows per-CU behavior.

D. Agent Efficiency

Agent efficiency as relating to wait states, number of REDIS calls, batch/bulk operation, number of threads, agent slot occupancy rates, pull model.

1) Thread Impact Analysis: Four threads is optimal... discover why this is.

2) CU-Agent Occupancy Rates: Discuss how "full" pilots are with regards to actively-executing CUs, reasons why this could be the case, tie

E. Single-Resource Multi-Pilot Analysis

Figure6 illustrates the impact of multiple pilots on BigJob performance vs a single pilot with the same total core count on a single resource.

1) Additional pilots allow inefficiencies caused by large pilots to be ameliorated by distributing the load from a single large pilot to several smaller pilots. However, when an Pilot is small enough that these inefficencies are not an issue, adding additional Pilots does not increase performance, as figure**??** shows.
2) There appears to be some overhead with regards to CU throughput and execution time for larger numbers of pilots (evident with 4 pilots for smaller BJ sizes, see figures**??**,**??**).

F. Multi-Resource Multi-Pilot Analysis

G. Mathematical Modeling

There are two modelling scenarios to consider; first, that of an ideal Pilot-Job system. Second, an actual case-study courtesy of BigJob operating under real-world conditions.

1) Idealized Pilot-Job Modelling: For an idealized Pilot-Job, we make several assumptions. We assume an idealized resource arrangement, in that resource queuing times are always zero and that there are an infinite number of cores waiting to be used. If multiple resources are to be used, we assume that there is zero latency between resources, and that all resource capabilities are homogeneous. We also assume that all N CUs are homogeneous.

Given A Pilots of C cores each, each Pilot has a startup overhead of A_{so} seconds before CUs begin to be executed. We consider this to be the fixed Pilot startup costs α, which cannot be influenced by adding additional Pilots or additional cores.

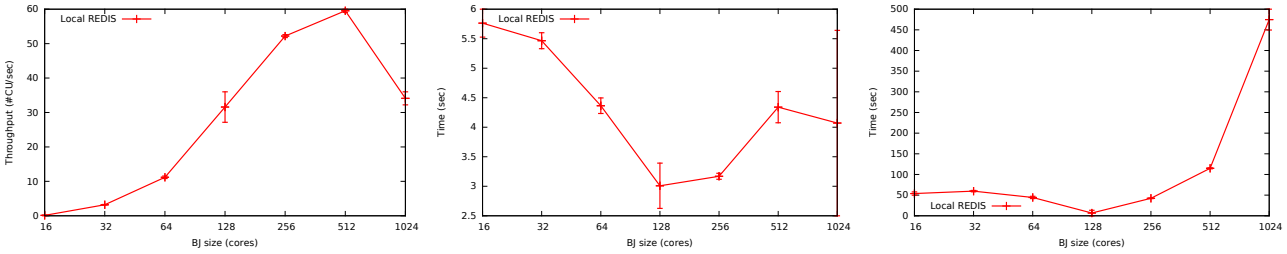
Each Pilot takes A_{qo} seconds to process an incoming CU and place it into the queue. Each Pilot takes A_{eo} seconds to remove a CU from a queue and begin executing. Finally, each Pilot takes A_{po} seconds to purge a finished CU. Variables A_{qo}, A_{eo}, and A_{po} may be combined into a single variable representing the time in seconds required for the Pilot to process a single CU, β.

We consider that there is a single key-value store used to store CU and pilot state information, with a fixed latency D_l. A Pilot must communicate with this key-value store to process CUs. The number of round-trips for an incoming CU to be placed onto an Pilot queue is represented as D_{qt}. The number of round-trips for a queued CU to be marked as executing is represented as D_{et}. The number of round-trips for an executed CU to be purged from the key-value store is D_{pt}. These round-trips may be combined and multiplied by the latency of a single round-trip D_l, giving the total time in seconds needed for the key-value store to process a single CU γ. There are additional round trips that could be considered which occur during the creation of Pilots roughly equivalent to the number of round trips needed to process a single CU– however, as the number of Pilots created is usually significantly less than the number of CUs, we do not consider these round trips in the model in order to reduce the model's complexity.

Each CU requires C_e seconds to execute its payload on a single core independent of agent/pilot overhead. We consider this time to be ϵ. Additional cores and Pilots can improve the execution time.

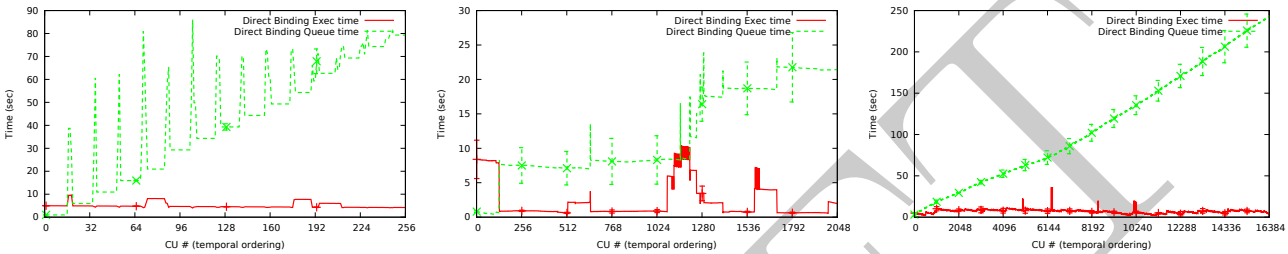
We may consider A multiple Pilots executing simultaneously on either local or distributed resources. We consider the key-value store latency D_l to be constant for every pilot, regardless of location. Multiple Pilots are assumed to split all CUs cleanly between them, and to all be available to begin execution simultaneously.

Given this model, we are then left with four primary components of the Pilot-Job execution model – the indivisible Pilot startup overhead α, the time it takes for the Pilot to manipulate a single CU β, the time lost in latency due to key-value store round trips γ, and the CPU time required to actually process the CU's workload ϵ. To review, they are calculated as



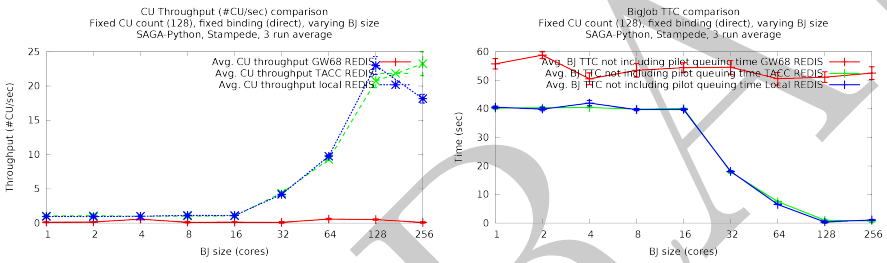
(a) Average CU throughput (b) Average CU execution time (c) Average BJ TTC

Figure 1. Single-Pilot BigJob experimentation with weak scaling on Stampede. The number of CUs is $16BJ_{size}$. CUs which should take zero time to execute ("sleep 0") are used. Four threads per agent are used as is standard (as of 9/23/2013), making this a "default" BigJob run. Direct binding is used. Local (TACC) REDIS is used.



(a) 16-core BJ (b) 128-core BJ (c) 1024-core BJ

Figure 2. Single-Pilot BigJob experimentation with weak scaling on Stampede. The number of CUs is $16BJ_{size}$. CUs which should take zero time to execute ("sleep 0") are used. The X-Axis corresponds to individual CUs, temporally ordered. Local (TACC) REDIS is used. Direct binding is used.



(a) Average CU throughput (b) Average BJ TTC

Figure 3. Single-Pilot BigJob experimentation with a Fixed CU count (128) on Stampede. CUs which should take zero time to execute ("sleep 0") are used. Different REDIS servers are experimented with: Local (headnode), TACC (still "local"), GW68 (remote).

follows:

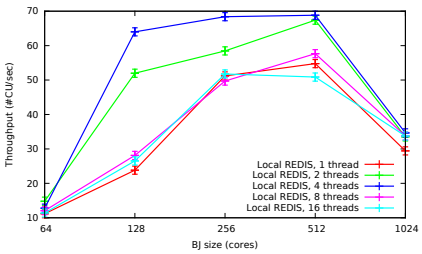
$$\begin{aligned} \alpha &= A_{so} && \text{Pilot startup overhead} \\ \beta &= A_{qo} + A_{eo} + A_{po} && \text{Single CU processing time, Pilot} \\ \gamma &= (D_{qt} + D_{et} + D_{pt})\, D_l && \text{Single CU processing time, database} \\ \epsilon &= C_e && \text{Single CU processing time, CPU} \end{aligned} \tag{1}$$

To model the time for a Pilot-Job system to complete execution on a single-core system, the Pilot processing costs are multiplied by the number of CUs N and added to the fixed Pilot startup overhead as follows:

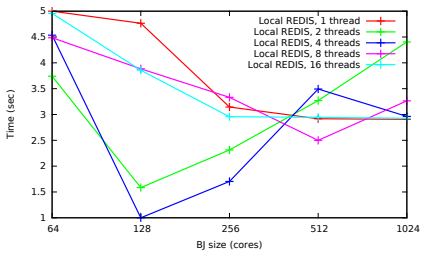
$$\begin{aligned} \alpha+ & && \text{Pilot startup overhead} \\ N\beta+ & && \text{Pilot CU processing overhead} \\ N\gamma+ & && \text{Database communications overhead} \\ N\epsilon+ & && \text{CU CPU processing time} \\ = T(N) & && \text{Execution time in seconds, single-core} \end{aligned} \tag{2}$$

Extending the model to make use of multiple cores C modifies only term ϵ, as the Pilot startup overhead, Pilot CU processing overhead, and database communication overhead are not reduced by the addition of cores due to the assumption of a single-threaded Pilot-Manager:

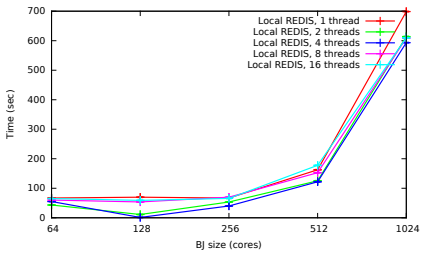
$$T(N, C) = \alpha + N\beta + N\gamma + \frac{N\epsilon}{C} \tag{3}$$



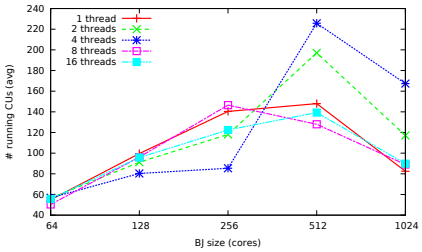
(a) Average CU throughput



(b) Average CU execution time

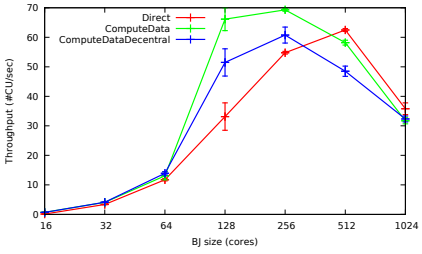


(c) Average BJ TTC

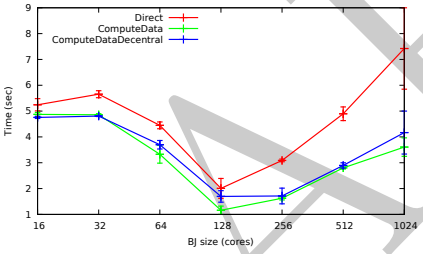


(d) Average Active CUs

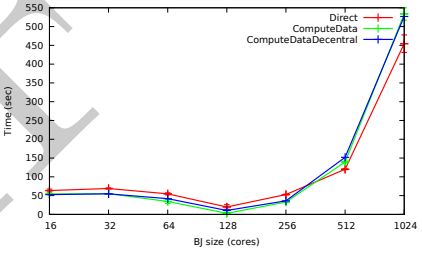
Figure 4. Single-Pilot BigJob experimentation with weak scaling on Stampede, with varying numbers of threads per agent. The number of CUs is $16BJ_{size}$. CUs which should take zero time to execute ("sleep 0") are used. Local (TACC) REDIS is used. Direct binding is used.



(a) Average CU throughput



(b) Average CU execution time



(c) Average BJ TTC

Figure 5. Single-Pilot BigJob experimentation with weak scaling on Stampede, demonstrating the performance of compute-data services. The number of CUs is $16BJ_{size}$. CUs which should take zero time to execute ("sleep 0") are used. Local (TACC) REDIS is used. Direct binding is used.

Considering multiple Pilots A proportionally reduces the Pilot CU processing and database communication overheads through parallel processing of CUs and communication with the database:

$$T(N,C,A) = \alpha + \frac{N\beta + N\gamma}{A} + \frac{N\epsilon}{AC} \quad (4)$$

Weak scaling variation:

$$T(N,C,A) = \alpha + \frac{CN\beta + CN\gamma}{A} + \frac{CN\epsilon}{AC} \quad (5)$$

2) BigJob Modelling: Adapting the ideal Pilot-Job model to model BigJob focuses mainly on adding additional sources of overhead and speedup as identified in previous subsections. The primary two modifications to the model required are a overhead term to model the increased overhead generated by additional cores and the consideration of a multi-threaded BigJob Pilot.

Should we consider the role of scaling overhead, after an analysis of BigJob's master-worker architecture we determine that the scaling behavior is linearly proportional to the number of cores. The degradation of performance is represented by variable χ.

The role of a multithreaded agent with additional threads t can be considered to effectively add additional agents. There is some overhead introduced by thread synchronization/locking/resource contention, but we do not consider this. Experimentally, observations show that overhead from adding additional threads becomes a net loss after about 4 threads.

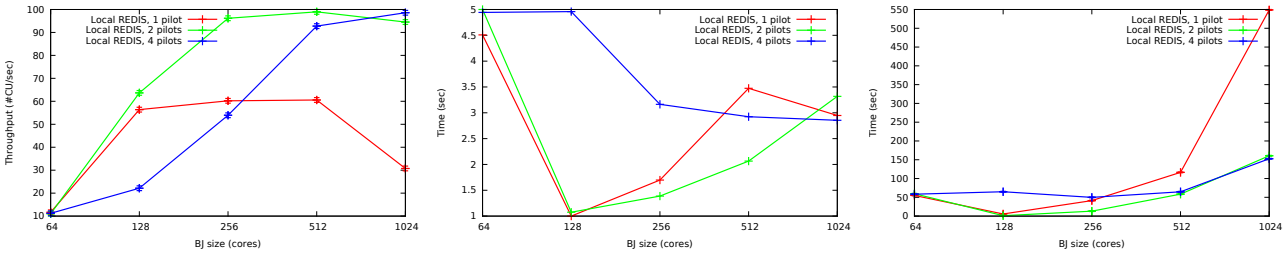
$$T(C,N,A,t) = \alpha + \frac{N\beta + N\gamma}{At} + \frac{N\epsilon}{AtC} + AC\chi \quad (6)$$

Figure 10 is used to reason out some of the BigJob inputs for the general Pilot performance model. Figure 12 shows the model with BigJob variables.

[1]

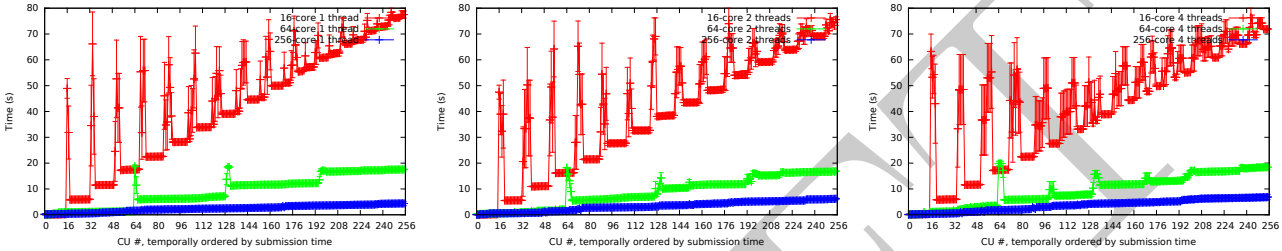
REFERENCES

[1] Andre Luckow, Mark Santcroos, Andre Merzky, Ole Weidner, Pradeep Mantha, and Shantenu Jha. P*: A Model of Pilot-Abstractions. In *8th IEEE International Conference on e-Science 2012*, 2012.



(a) Average CU throughput (b) Average CU execution time (c) Average BJ TTC

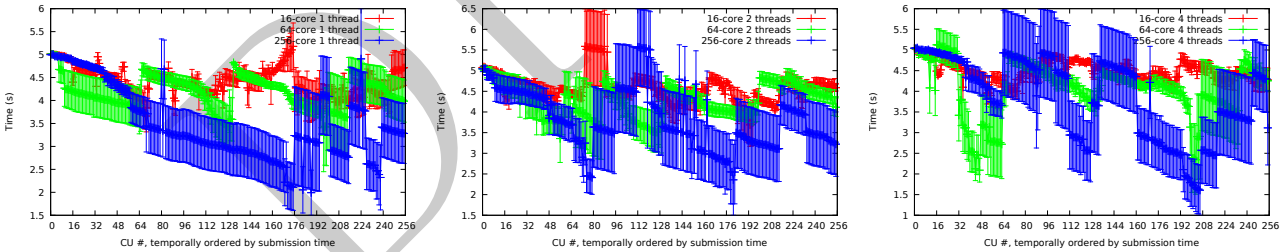
Figure 6. Multi-Pilot BigJob experimentation with weak scaling on Stampede. CUs which should take zero time to execute ("sleep 0") are used. The total number of CUs across all pilots is $16BJ_{size}$, with CUs evenly distributed between pilots. The size of each pilot is equivalent to $BJ_{size}/NUMPILOTS$. The number of threads per pilot is 4. Direct binding is used. Local (TACC) REDIS is used.



(a) Per-CU queuing time, 1 thread (b) Per-CU queuing time, 2 thread (c) Per-CU queuing time, 4 thread

(d) Per-CU queuing time, 8 thread (e) Per-CU queuing time, 16 thread

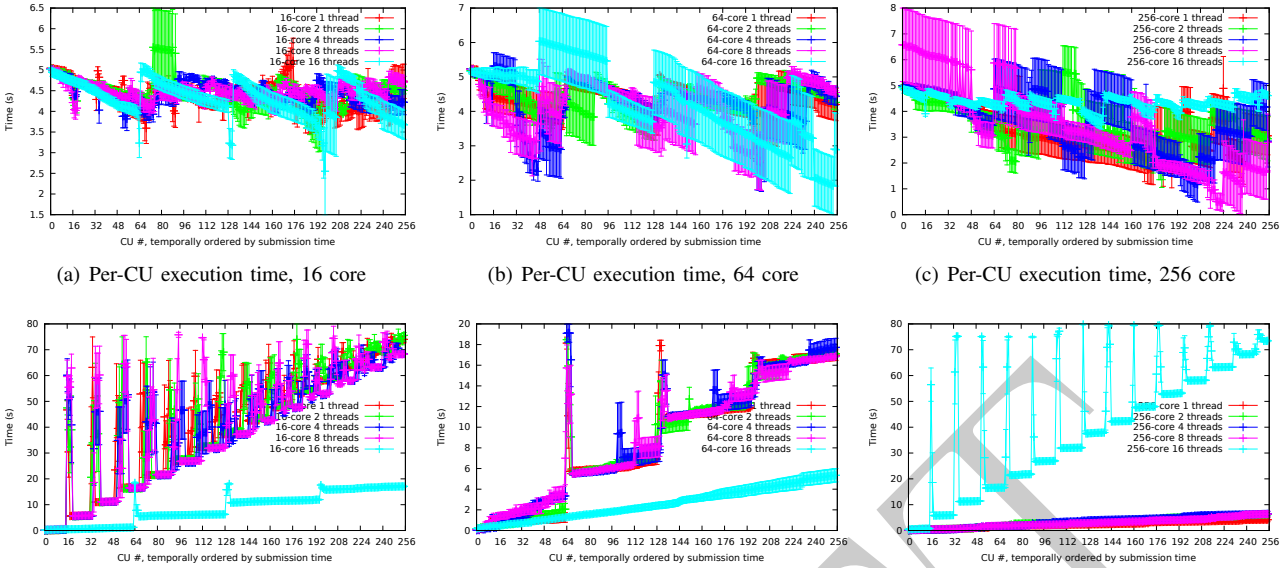
Figure 7. Per-CU queuing time, threads fixed



(a) Per-CU execution time, 1 thread (b) Per-CU execution time, 2 thread (c) Per-CU execution time, 4 thread

(d) Per-CU execution time, 8 thread (e) Per-CU execution time, 16 thread

Figure 8. Per-CU execution time, threads fixed



(a) Per-CU execution time, 16 core

(b) Per-CU execution time, 64 core

(c) Per-CU execution time, 256 core

(d) Per-CU queuing time, 16 core

(e) Per-CU queuing time, 64 core

(f) Per-CU queuing time, 256 core

Figure 9. Per-CU execution and queuing times, BJ size fixed

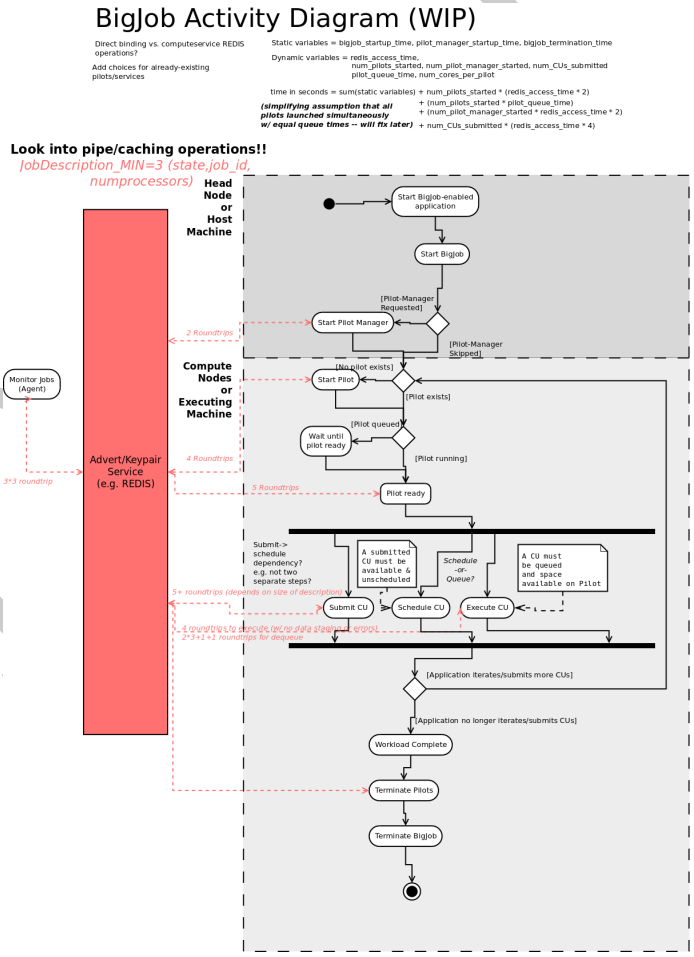
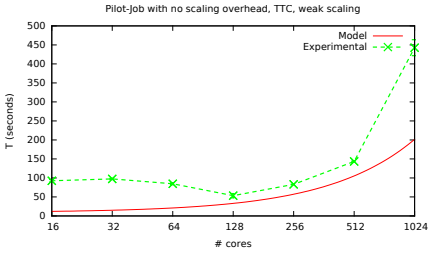
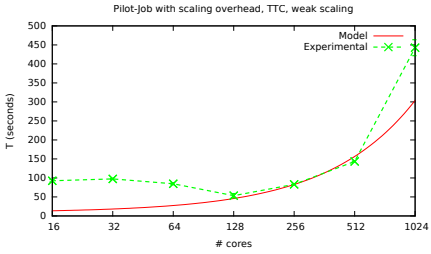


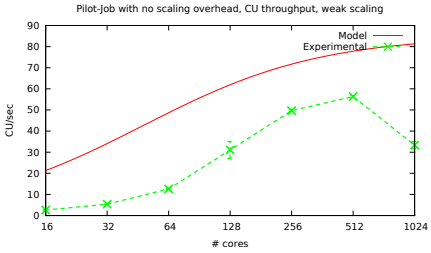
Figure 10. BJ Diagram



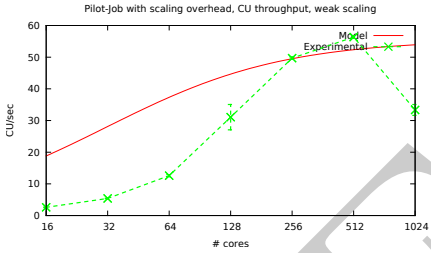
(a) Modelled BJ TTC (no overhead)



(b) Modelled BJ TTC (overhead)

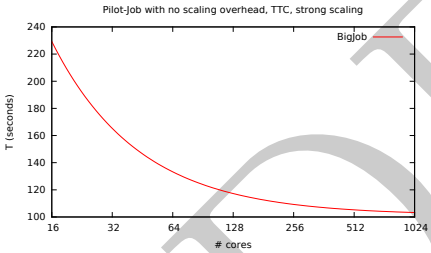


(c) Modelled BJ CU throughput/sec (no overhead)

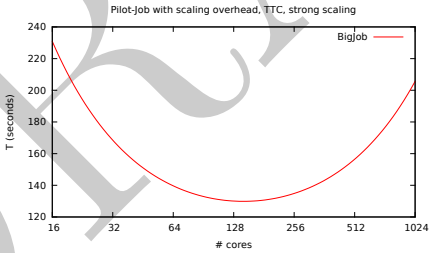


(d) Modelled BJ CU throughput/sec (overhead)

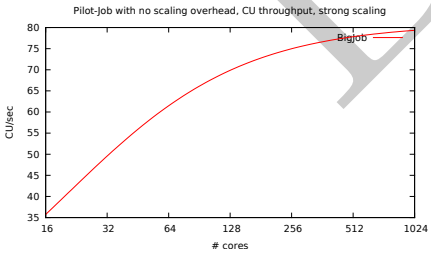
Figure 11. Modelling plots, weak scaling.
$a_{so} = 5, n = 16C, d_{qt} = 5, d_l = .001144, a_{qo} = .2, d_{et} = 4, a_{eo} = 2, d_{pt} = 8, a_{po} = .2, c_e = 2, \chi = .10$



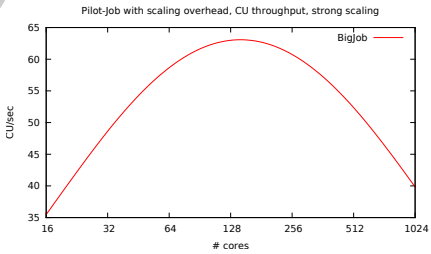
(a) Modelled BJ TTC (no overhead)



(b) Modelled BJ TTC (overhead)



(c) Modelled BJ CU throughput/sec (no overhead)



(d) Modelled BJ CU throughput/sec (overhead)

Figure 12. Modelling plots, strong scaling.
$a_{so} = 5, n = 8192, d_{qt} = 5, d_l = .001144, a_{qo} = .2, d_{et} = 4, a_{eo} = 2, d_{pt} = 8, a_{po} = .2, c_e = 2, \chi = .10$