

# Automating Collection of Performance Statistics on Various Cluster Implementations

Samuel Leeman-Munk, Aaron Weeden, Andrew Fitz Gibbon, Bradley Johnson-Stalhut, Matt Edlefsen, Gus Schuerger, David Joiner, Charlie Peck

Earlham College - [leemasa, amweeden06, fitz, carrick, edlefma, gaschue08, charliep] @ cs.earlham.edu , Kean University - djoiner@kean.edu

Blue Waters is a petascale computational resource coming on-line in 2012 at the National Center for Supercomputing Applications. This machine will be capable of sustained petaflop performance, that is over  $10^{15}$  floating point operations per second, significantly more than any other known computational resource in our galaxy. Our group is participating in an on-going project to explore the scalability of common N-Body software to petascale computational levels. This involves both the core algorithms and their implementation on hardware platforms with multiple levels of shared and distributed memory hierarchies.

In order to further this exploration we have designed software to allow the convenient collection of performance data over several different parallelism paradigms at different scales of parallelization and the subsequent collection of said data into visualization. Designed to be a kit for running on petascale resources, we have lovingly dubbed our software "PetaKit."

This work is supported by the National Science Foundation's Blue Waters Petascale Computing project, the Shodor Foundation, the SC09 Education Program and Earlham College.

## Technical Overview

### Configuration

The PetaKit uses GNU's autotools for configuration on the various computational resources. A script, `configure.ac`, checks environment variables to detect on which of the supported computation resources it is being configured. The PetaKit runs this script, specifying the desired style of parallelism (shared memory, distributed memory, or a hybrid of the two), which autotools uses to build the programs via whichever compiler is specified for the target resource in `configure.ac`. This produces a separate executable for each of the styles of parallelism for each of the programs that can be used by the harvester.

### Test Execution and Data Collection

The harvester, `stat.pl`, serves two roles – first it sets up the program's runs, then it collects the data and sends it to the centralized database. `Stat.pl` takes as arguments sets of parameters, such as style of parallelization and degree of parallelization (number of cores), and iterates over the applicable combinations of these parameters, producing a shell script for each. `Stat.pl` then sends these scripts to the appropriate scheduler.

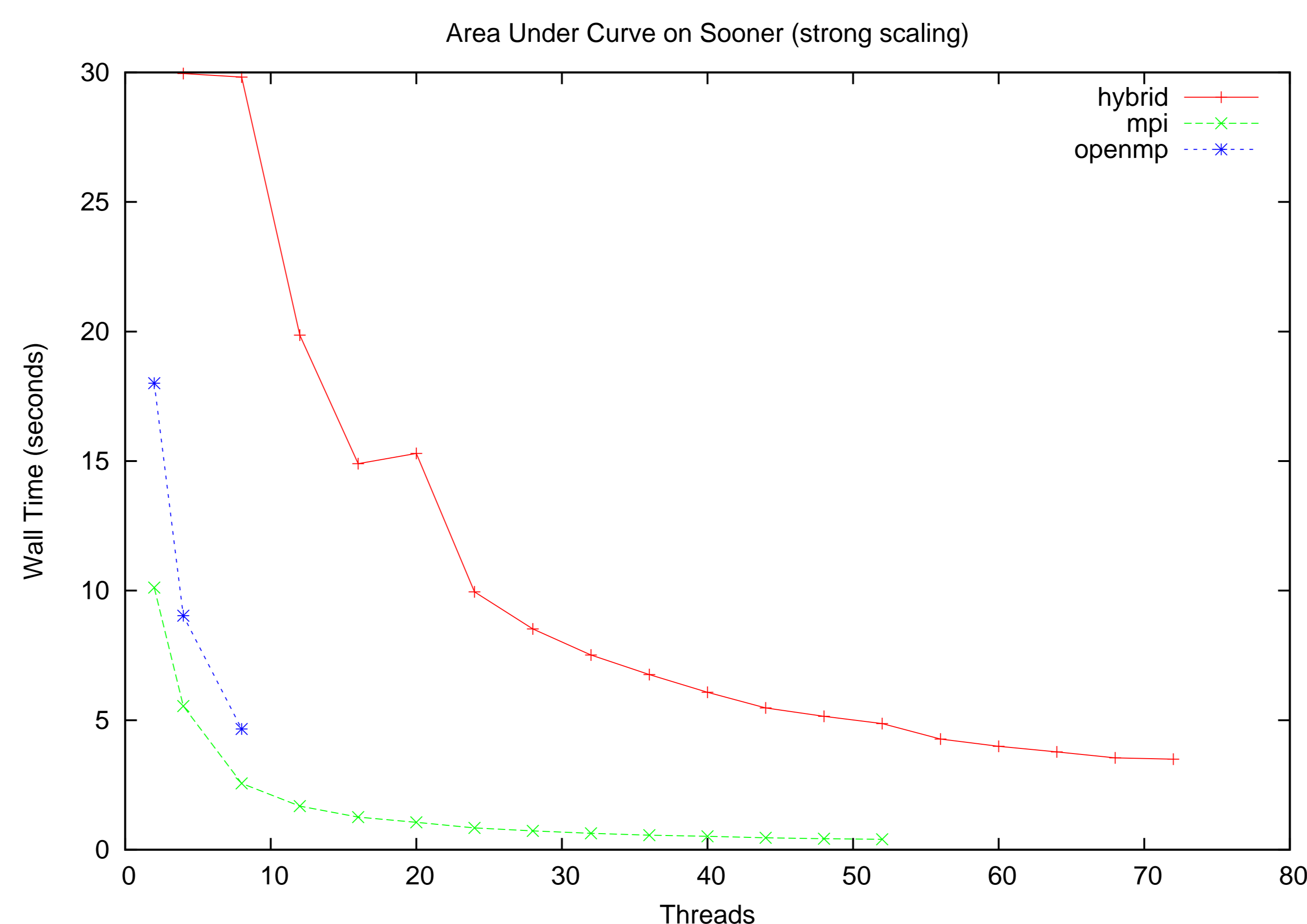
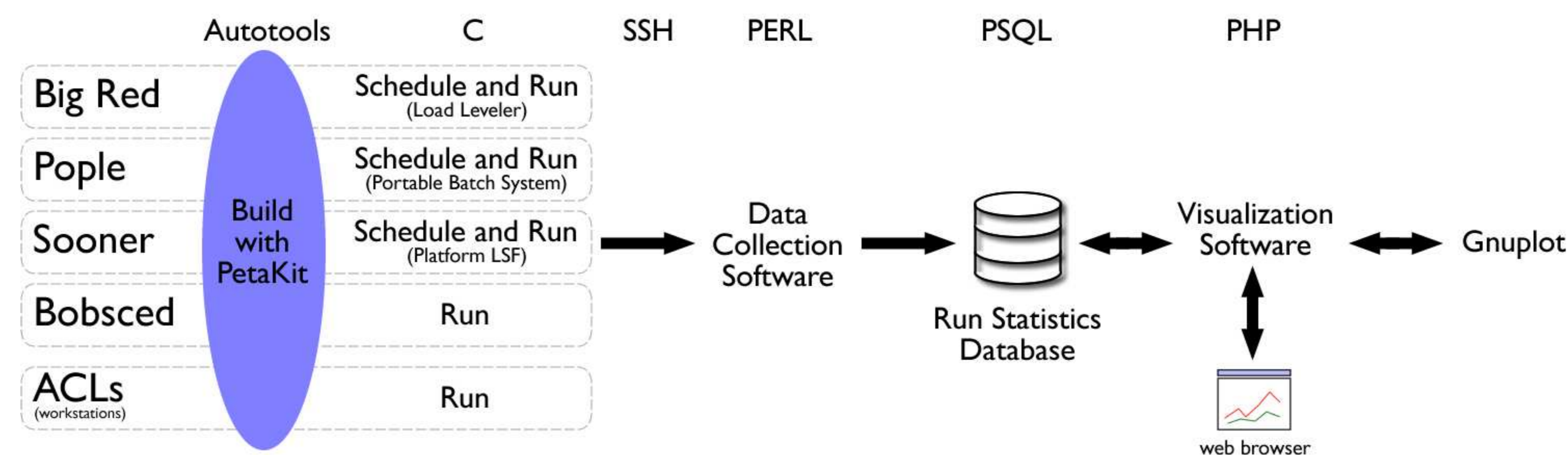
Each script that `stat.pl` constructs collects the performance data for the program it ran, as well as the parameters under which it was run. It then sends this data to another script, residing on the machine hosting the database, which formats it and stores it.

### Visualization

PetaKit includes a database visualization tool: a web based interface that allows users to interactively build a graph of run statistics by specifying and refining its parameters. The driving program, written in PHP, provides UI elements and parses user input. Using a combination of user provided information and run statistics retrieved from the PetaKit database, a data file and Gnuplot script are created on the fly, allowing for dynamic graph creation and data visualization.

### Future Work

At the moment our configurations are on a per-cluster basis. The software dynamically detects on what cluster it's running and sets its environment variables to work for that. We will want to expand our software to dynamically detect the individual variables and configure itself even on a machine it has never seen before. Then it will truly be portable. Similarly the run software relies on the user to tell it the specifics of the system, such as the scheduler to which to submit its runs. Automating this will further the user-friendliness of our system.



A performance statistics graph produced with PetaKit

## Design

### Programs

At the moment, PetaKit supports two programs:

- Area-under-curve: an embarrassingly parallel Reimann sum algorithm
- GalaxSee: a popular instance of the n-body problem

### Computational Resources

The PetaKit currently supports data collection on the following large-scale computational resources:

- Sooner – Teragrid resource at the University of Oklahoma. PowerEdge 1950, Xeon E54xx, 2 GHz, Infiniband
- Big Red – Teragrid resource at Indiana University. BladeCenter JS21 Cluster, PPC 970, 2.5 GHz, Myrinet
- Pople – Teragrid resource at Pittsburgh Supercomputing Center. SGI Altix 4700, Itanium 2 Montvale 9130M, 1.66 GHz, NUMalink

### How to Use PetaKit

The PetaKit project aims to make the genuinely complex process of building and running a set of parallel programs on an arbitrary cluster as easy as possible. At the moment PetaKit is not available to the public, but when PetaKit is released a user wishing to run a program it supports on a given cluster follows the following four steps:

1. Download and decompress the PetaKit package for the given software
2. Enter the created directory and run the `configure` script, which compiles the program for the given system's architecture
  - Four versions of the program are created: serial, shared memory, distributed memory, and hybrid shared-distributed
3. Run the included Perl script, `stat.pl`, specifying the parameters to use and the range of processors over which to run
  - `stat.pl` takes as a parameter the scheduler to use
  - The results are automatically sent to a centralized database via `ssh`
4. After the runs have completed, view the results on the browser-based visualizer.

