3X: A Data Management System for Computational Experiments

Demonstration Proposal

Jaeho Shin Andreas Paepcke Jennifer Widom
Stanford University
{netj,paepcke,widom}@cs.stanford.edu

ABSTRACT

3X, which stands for *eXecuting eXploratory eXperiments*, is a software tool to ease the burden of conducting computational experiments. 3X provides a standard yet configurable structure to execute a wide variety of experiments in a systematic way. 3X organizes the code, inputs, and outputs for an experiment, records results, and lets users visualize result data in a variety of ways. Its interface allows further runs of the experiment to be driven interactively. Our demonstration will illustrate how 3X eases the process of conducting computational experiments, using two complementary examples designed to quickly show the many features of 3X.

1. MOTIVATION

Organizing the data that goes into and comes out of experiments, as well as maintaining the infrastructure for running experiments, is generally a tedious and mundane task. Frequently one ends up in an environment improvised and hard-coded for each experiment. The problem is exacerbated when experiment runs must be performed iteratively for exploring a large parameter space. The goal of our system, 3X (for *eXecuting eXploratory eXperiments*), is to ease this burden, enabling users to more quickly make interesting discoveries from their computational experiments.

Computational and data-driven approaches have long been a standard method for scientific experimentation in many domains, and we see a growing number of fields depending on computational experiments. More generally, *data scientists* are emerging across many enterprises, wanting to ask questions in the form of computational experiments, and to discover new facts from the "big data" they have accumulated. We believe 3X in its current form can already considerably simplify and amplify the work associated with computational experiments. Our eventual goal is for 3X to act as a powerful "assistant" for users to explore the behavior of programs and large parameter spaces easily and effectively.

To put 3X in the context of other systems, the main focus of 3X is to ease the burden on the experimenter, in contrast to cluster management systems, such as *GridEngine* [2], *HTCondor* [8], *Mesos* [4], *PBS* [3], *SLURM* [9], and *Torque* [6], which aim to achieve higher performance and better resource utilization out of a cluster of machines. Systems such as *Tableau* [7] are focused on data visualization specifically. Although 3X includes some basic data visualization features, it is more focused on managing the experimental set-up and executing the variety of runs to produce the data. When sophisticated renderings of output data are required, 3X could be used in conjunction with Tableau or other visualization systems.

The 3X system as described in this demonstration proposal is fully implemented and functional, and is available as open source at `http://j.mp/to-3X` [5]. An extensive tutorial is included in the release. Please give 3X a try!

2. FEATURES OF 3X

Figure 1 gives a visual overview of 3X, showing how users set up experiments, and summarizing the features 3X provides. (The specifics in *Experiment Code*, *Input Parameters*, and *Desired Outputs* refer to our running example introduced shortly.) The primary three modes of 3X, described in more detail in the subsections below, are for: (1) organizing experiments; (2) running them; and (3) exploring and analyzing their results. These modes are supported in 3X by:

- A **standardized experiment repository**, where users organize the code of their experiments along with input data and output definitions in a systematic fashion, and where 3X records the execution results, again in a systematic (and storage-efficient) way.
- **Features for running the experiments**: to plan, control, and manage executions of multiple runs with varying input values on different target environments. 3X is agnostic to the user's programming language or system, and can execute runs on a user's local machine, on remote servers, and in parallel on compute clusters.
- **Immediate, interactive visualizations** of execution results, as tables and charts, to let users explore and analyze the data as soon as their computations finish. Without writing or running any scripts or queries, users can see results as soon as the runs finish. Records in tables and data points in charts are linked to the runs that produced them, so users can easily drill-down to the input parameters corresponding to interesting output.

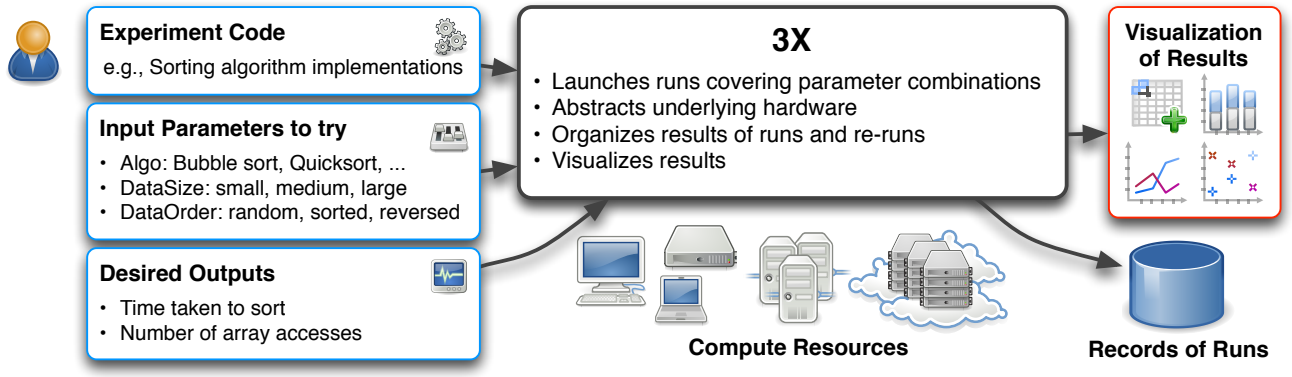

Figure 1: High-level view of features provided by 3X.

2.1 Creating and Running an Experiment

We illustrate user interactions with 3X through a layman's example that still serves to demonstrate many of its features. Suppose an experimenter wishes to explore the relative behaviors of five sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Quicksort, and Merge Sort. The algorithms are implemented in the *experiment code* shown in the top-left box of Figure 1. The experiment is to see how each algorithm behaves on three input sizes. Additionally, for each load size, the experiment calls for trials that have the respective data either randomly unsorted, already sorted, or in reverse sort order. We call these three independent variables—the choice of algorithm, the load size, and the sortedness of the data—the *input parameters* to 3X (middle-left box of Figure 1). For each, or optionally some, combination of input parameter settings, the sorting program observes the time it took to execute, and the number of data accesses. We call these values the 3X *output parameters* (bottom-left box of Figure 1). While the user obviously needs to write the sorting program, 3X provides conventions for how 3X starts the program, and how the program delivers results back to 3X. The only remaining step for the user is to select a desired *compute resource*. 3X can run experiments sequentially or in parallel, using laptops, desktops, servers with multiple cores (local or remote), and remote compute clusters.

3X now automatically executes all requested input variations, keeps track of the computation results, and manages computation or communication failures. Through the 3X graphical user interface (GUI), the user can observe the processing of the various input combinations, referred to as *runs*. Figure 2 shows a screenshot of what 3X shows the user during the running phase of the experiment. If the user wishes to repeat one or more runs, 3X keeps track of the re-runs and all of their outputs.

Finally, the user needs to instruct 3X how to process experiment outputs before displaying them. Currently there are three options: (1) specify a regular expression that scans the standard output/error and files to extract a piece of text; (2) provide the name of an output file, which might contain a custom plot or a visualization of higher dimensional data; or (3) provide another program, called the *extractor*, which performs custom processing on the experiment program's output (when output processing is too specialized for either of the previous options).

2.2 Exploration and Analysis of Results

Experiment results can be visualized as soon as the first few runs finish execution. Visualizations provided by 3X are more powerful than static charts of the result data, because they are linked to *provenance* information: Users can drill down through the result visualizations to see the particular runs and specific input values that produced data points of interest.

Figure 3 shows 3X results in tabular form, displaying the input values and output data of the runs as rows and columns. Users have control of which columns are visible in the table, how they are ordered, by which input parameters the rows in the table are grouped, what aggregate statistics of grouped rows are shown for each column, and any conditions rows must satisfy to be visible. Multiple aggregate statistics for an output variable can be shown at the same time, currently including *mean*, *standard deviation*, *median*, *min*, *max*, *mode*, and *count*, with the variable's data type determining which statistics are available. Any aggregate cell in the table can be inspected to trace the individual runs that contributed to it. If a user wants to augment some of the rows with data from new runs, the additional runs or re-runs can be scheduled directly from the table.

Figure 4 shows 3X results in chart form, displaying the data shown in the table as a two-dimensional plot. Multiple input and output parameters of interest can be chosen to automatically create a chart based on the types and ranges of the data. Currently, 3X supports only line charts and scatter plots with simple options, but we will soon extend to a wider variety of charts and features. Importantly, each visual element in 3X's automatically-generated charts is interactive, inviting users to inspect which runs and input values produced data points of interest. This feature can be seen in the "popover" boxes in Figure 4.

3. IMPLEMENTATION OF 3X

Figure 5 shows the logical architecture of 3X, which consists of three main components: the *User Layer*, the *System Layer*, and the *Experiment Repository*. The graphical user interface (GUI) to the user layer exposes the main 3X facilities to users as a variety of interactive views. The GUI is used to monitor runs, visualize results, and request re-runs. The *Experiment Repository* manages the data relevant to an experiment. The repository stores records of runs and the

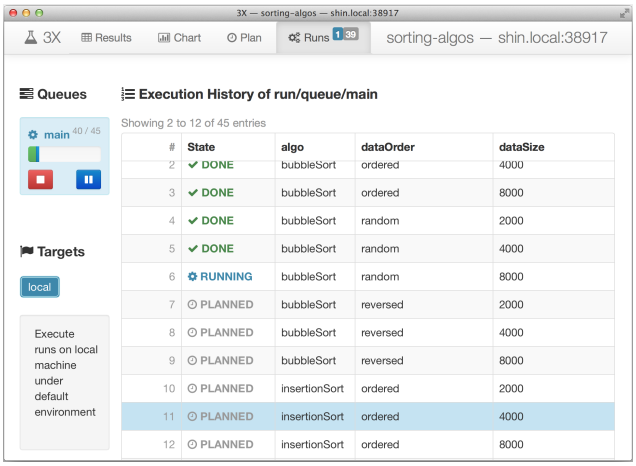

Figure 2: Monitoring the progress of experiment runs.

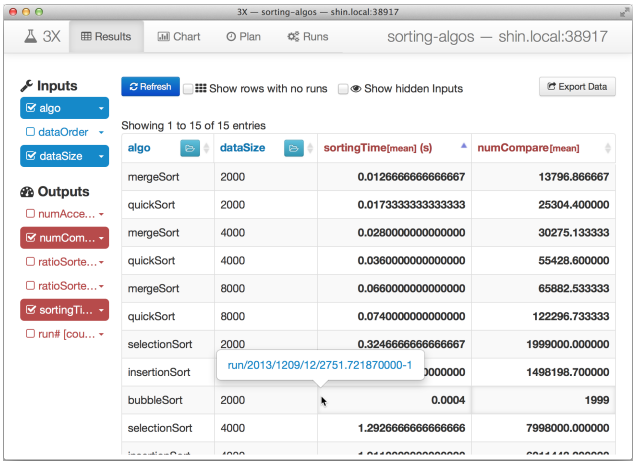

Figure 3: Tabular result data.

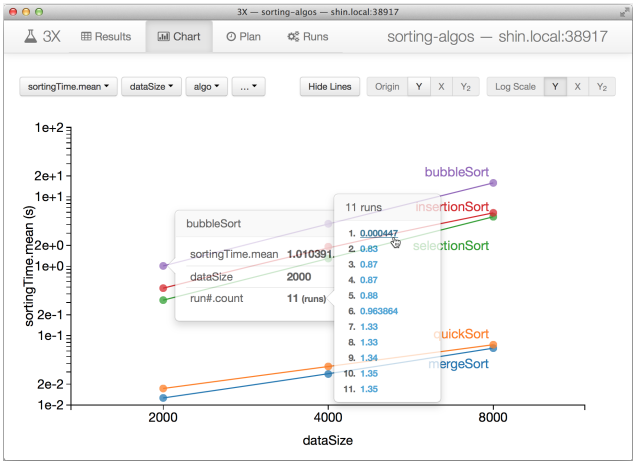

Figure 4: Charted result data.

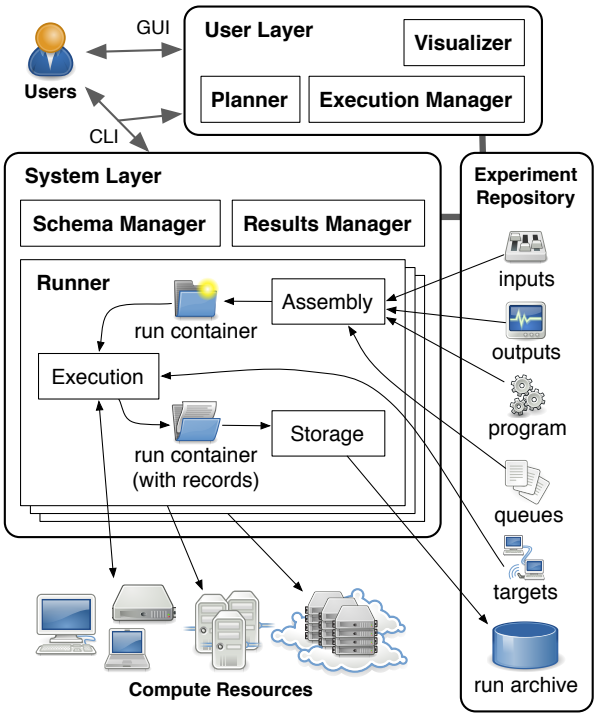

Figure 5: Architecture of the 3X system.

target configurations for execution, as well as the experiment code, the input parameters, and the output parameter definitions. The *System Layer* provides the core 3X execution facilities for controlling the underlying compute resources, described in more detail below, as well as facilities for defining the schema of the repository for each experiment, and performing operations on the repository.

In addition to the GUI, 3X also has a command-line interface (CLI), which allows advanced users to interact with 3X more directly; currently the CLI is also used for initial creation of experiments. The 3X CLI has been designed to be scriptable and extensible: users can extend 3X, or operate it from within their own applications. For example, if an analyst wishes to run machine-learning algorithms over periodically-collected data, the 3X CLI could be instrumented to automate the task. As another example, suppose a set of users wish to run experiments on a new compute cluster with a customized job scheduler. Adding new compute resources is supported by extending or overriding the behavior of existing 3X "runners", explained next.

Space constraints preclude a detailed implementation description for most of the 3X system. In the rest of this section we discuss the *Runner* subcomponent of the system layer (Figure 5) in a bit more detail, since it is one of the most important parts of our implementation. For each type of target execution environment 3X supports, there is a runner that provides an abstraction of the actual actions necessary for executing runs at the target. As each run is assigned to the responsible runner, 3X creates a *run container* that encapsulates all of the data that goes into and comes out of the program for the run. The runner proceeds through the following stages:

1. **Assembly.** The container for a run is first *assembled*

by copying into the container the experiment program and any user-defined data files relevant to the chosen values of the input parameters. For cluster-type targets that execute runs on multiple machines, 3X performs assembly separately on each machine, to avoid any bottlenecks.

2. **Execution.** The experiment program is executed under a controlled and sanitized environment at the target. Records of the execution, such as the output, generated files, exit status, CPU time, and other common resource usage profiles, are captured under the run container. After the execution finishes, 3X performs the user-specified output processing for each output parameter (recall Section 2.1), in order to set its value in the run container.
3. **Storage.** Finally, 3X finishes the run by storing the run container in the experiment repository. Data for the output parameters is indexed automatically for faster access. Since it is common for identical copies of files to be produced across many runs, 3X detects duplicates and merges them into a single copy to minimize storage overhead.

4. DEMONSTRATION WALK-THROUGH

In the live demonstration, we will show the features of 3X using two examples: the simple sorting algorithms study that was used as a running example in Section 2, and a study exploring connectedness behavior in random networks.

4.1 Sorting Algorithms Study

Most conference attendees will have studied the theoretical complexity of sorting algorithms in their basic Computer Science classes, but few will have carried out experiments over a variety of data sets and visualized the comparative results. We will walk visitors through setting up the sort experiment described in Section 2.1. Depending on network connectivity, we will either have 3X perform the experiment runs locally, or remotely in parallel on our lab's cluster. Through the 3X GUI, visitors will observe results arriving asynchronously and being made available for immediate exploration. While computations are ongoing, we will visualize the results using charts that demonstrate how the different algorithms behave under the different experimental conditions. If all goes well, visitors will have questions that prompt us to explore the provenance of different data points in the charts.

One example we will show is a line chart of running time versus data size for each algorithm. We will confirm that Quicksort, whose worst case time complexity is $O(n^2)$, has comparable performance in average case to Merge Sort, which is an $O(n \log n)$ algorithm. Then, using a dot-plot of running time versus algorithm for each data size, we will observe a set of outliers for Insertion Sort and Bubble Sort, having extremely small running time. Drilling down on the outlier dots, we learn that all of them correspond to runs on data that was already sorted. This interaction raises the question of whether Insertion Sort and Bubble Sort would also be efficient for data that is mostly but not completely sorted. Modifying the 3X experiment to explore that question is left for visitors to do later on their own, if so inspired.

4.2 Giant Components in Random Networks

The sorting experiment generates only numerical outputs, so we will use a second example to demonstrate 3X's capability of handling image outputs. In network science, it is a well-known phenomenon that in random networks, a "giant component" connecting most of the vertices in the network quickly emerges when we increase the probability of an edge between each pair of vertices [1]. To see this effect, our experiment program takes as input a network size (in number of vertices), and an edge probability. The output of each run is an image file showing the generated network as a *node-link diagram*.

Numerical outputs have numerous standard statistics available for summarizing result values, but image outputs lack such standard summaries, and typically are browsed one at a time. 3X provides an *overlay* aggregate function for images, which takes a set of images, reduces their opacity, and renders them one on top of each other to produce an aggregate image. For our demonstration, we will pre-run a set of experiments for a variety of network sizes, with increasing edge probabilities for each size. Several runs for each setting are overlaid automatically by 3X to visualize smoothed effects on the rendered graphs. Through these overlaid aggregate images, visitors will be able to appreciate how easy it is to discern visually at which edge-probability values a giant component emerges for each network size.

5. REFERENCES

[1] F. R. K. Chung and L. Lu. *Complex graphs and networks*, chapter 6. The Rise of the Giant Component. American Mathematical Society, 2006.

[2] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. In *Proc. of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2001.

[3] R. L. Henderson and D. Tweten. Portable Batch System: External reference specification. Technical report, NASA, Ames Research Center, 1996.

[4] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proc. of the 8th USENIX conference on Networked systems design and implementation*, 2011.

[5] J. Shin. 3X Documentation: Tutorial with Examples. `http://netj.github.io/3x/docs/tutorial/`.

[6] G. Staples. TORQUE resource manager. In *Proc. of the 2006 ACM/IEEE conference on Supercomputing*, 2006.

[7] C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 2002.

[8] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience*, 2005.

[9] A. B. Yoo, M. A. Jette, and M. Grondona. SLURM: Simple Linux Utility for Resource Management. In *Proc. of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, 2003.