# Assignment #4: Data Collection and Preparation

Caitlin Ross & Noah Wolfe
02/18/2016

## 1 Data Source (ROSS)

In this assignment, we have decided to generate/collect data from our ROSS research projects. ROSS is a massively parallel discrete-event simulator that can process billions of events per second [3], [1]. ROSS models are made up of a collection of logical processes (LPs). Each LP models a distinct component of the system. LPs interact with one another through events in the form of time-stamped messages. An MPI task is abstracted as a processor element (PE) in ROSS. Each PE owns a number of LPs and schedules events in time-stamp order for all LPs assigned to it. Events that are destined for a logical process on another PE (i.e. remote events) are sent as MPI messages.

For parallel execution, ROSS uses an optimistic synchronization protocol. When an LP determines it has processed events out of timestamp order, it rolls back event computation and re-executes events in the correct order. The Global Virtual Time (GVT) is computed on a regular basis, which allows for a reclamation of events with timestamps smaller than GVT. The ROSS scheduler computes GVT every `gvt_interval` iterations in the scheduling loop. Another variable `batch` can be set to determine the number of events processed during each iteration of the scheduling loop, so `gvt_interval*batch` events are processed between successive GVT computations. Changing these parameters can trigger large changes in metrics such as number of reverse computations, remote events, event efficiency, etc. Currently, the only data collection is averaged over all processes and collected at the very end of the simulation.

## 2 Research Questions

ROSS is designed to be a very fast and efficient simulator. In order to get the best possible speed-up, developers need to know what is going on behind the curtains throughout the simulation. We need to know not only if a simulation is experiencing hotspots, but also be able to detect the cause of those hotspots. This requires collection of data throughout the simulation. Two research questions that can be solved by analyzing ROSS runtime system data are:

- Where is my simulation spending it's compute time?
- Which metrics are most influenced by the `batch` and `gvt_interval` parameters?

The first research question should be answerable by simply visualizing the large set of collected data. The second item will require much more data manipulation and analysis such as a weighting system to find correlations between ROSS parameters and simulation speed.

## 3 Hypotheses

The results to the above research questions are very dependent on the given model so we will focus on using the Slim Fly network model which simulates a large cluster of compute nodes connected and communicating in a Slim Fly topology. Simulating a uniform random workload and using the minimal routing algorithm, we expect to see the majority of the simulation compute time being spent performing forward events. This hypothesis comes from prior knowledge of the mapping of LPs to PEs for the Slim Fly model in ROSS. A uniform distribution of compute nodes, routers, and MPI processes means work should be evenly distributed and result in very little rollback activity. In the case of the second research question, we expect to find that the `batch` size parameter has the most influence on limiting remote events. This prediction is based on prior experience playing with the batch size to minimize execution time.

# 4 Data Format, Extraction & Manipulation

Using the simple csv format, the raw data is structured so that the columns represent the various ROSS parameters and metrics and the rows represent those same parameters collected at a later instance of simulation time. The default sampling frequency is $0.01*$`simulation_end_time` but can be adjusted with a command line flag `--report-interval=`$n$ where $n \in \{0.999, ..., 1/simulation\_end\_time\}$. Each MPI process creates its own data file and appends the MPI rank number to the end of the filename. Filenames can be adjusted with the command line flag `--stats-filename=string`. We are currently collecting a total of 20 ROSS simulation metrics (20 columns) and can sample up to 50,000 points in time for 4 MPI processes. This configuration results in 4 million data points. Even larger data sets can be generated with longer simulation runtimes.

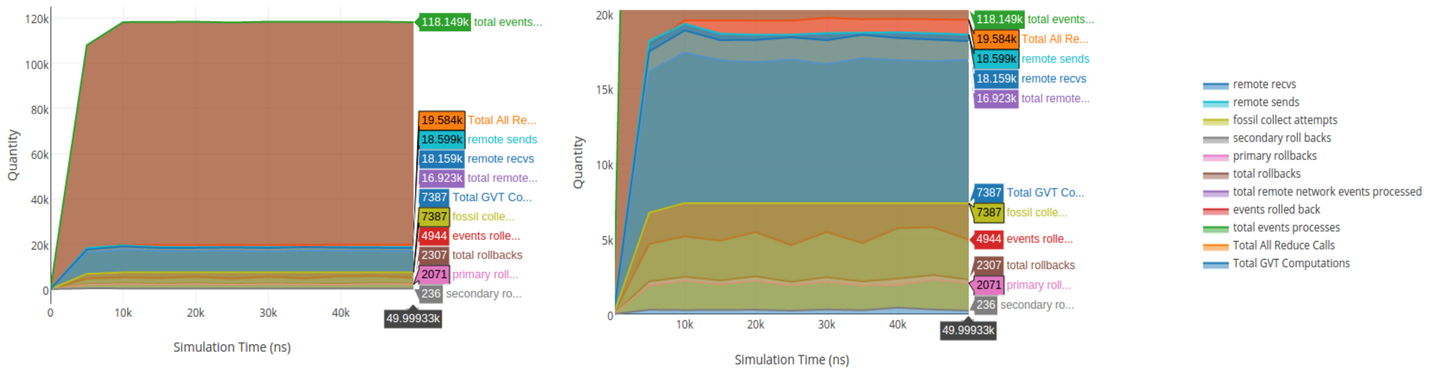
The results we show here are from Slim Fly model runs where we vary either the `batch` or `gvt_interval` parameters. We perform experiments where `batch` is held constant at 8 and `gvt_interval` varies from 2 to 1024 by powers of 2. We also perform experiments where `gvt_interval` is held constant at 8 and `batch` varies from 2 to 128 by powers of 2.

# 5 Sample Visualizations

The first visualization shown in Figure 1, was created using the Plotly web-based visualization tool and is used to answer the first research question posed in section 2. The figure displays the total events computed in the simulation as well as the number of events computed for specific event types such as rollbacks, fossil collection, etc. As can be seen in the figure, the number of All Reduce computations exceeds all other types of computation so it appears the majority of the time is spent performing MPI_All_Reduce.

To answer the second research question, we decided to visualize our data using a parallel coordinates graph. We decided to use d3 again for this, as we found a very easy to use parallel coordinates library [2] that is built on top of d3.js. The graphs are also hosted at `http://homepages.rpi.edu/~rossc3/parcoords.html`. We were able to use this d3.parcoords library to easily add interactive features, such as selecting a range on any number of axes as well as being able to shuffle the axes. Hovering over the axis names and scrolling with a mouse will also rotate the names so that they're easier to read. The first axis in both graphs show either the value for `batch` or `gvt_interval` and all other columns are the same between both graphs. Each line is a result for a GVT computation for a PE in a given simulation run. All lines for a given `batch` or `gvt_interval` value are given the same color. For example, in Figure 2, all data points, regardless of GVT computation and PE, are colored yellow for `batch=128`.

Figure 2 shows the parallel coordinates graph for the various `batch` values and Figure 3 shows the graph for various `gvt_interval` values. Some metrics had values of 0 for all data points, so we removed those metrics from the graphs, leaving 15 metrics. `batch` has the most influence on total number of events processed, events rolled back, primary and secondary roll backs, efficiency, and remote sends and receives. These metrics are interrelated. For instance, efficiency is based on the ratio of roll backs to total events. As there are more rollbacks, efficiency is lower because the simulation is spending more time processing reverse events over forward events. `gvt_interval` does not affect total events processed and number of events rolled back as much as the `batch` parameter (Note that the scales are different between the batch and gvt-interval graphs). Because of this, efficiency is less affected for the different `gvt_interval`.



(a) Overall View

(b) Dense Section Zoomed View

Figure 1: Visualizations of the ROSS data set showing the number of events computed in each section of the simulation over the length of the simulation. Sub-figure 1a shows a global view of all metrics while sub-figure 1b shows a view zoomed in on the dense region at the bottom of sub-figure 1a.

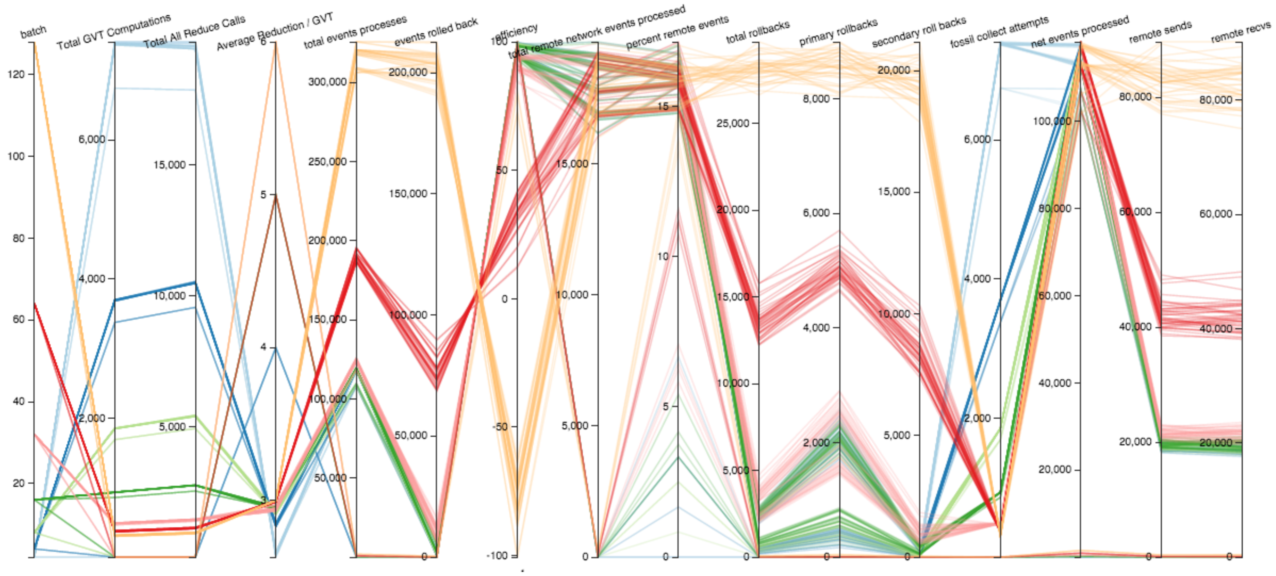


Figure 2: Parallel Coordinates graph for runs with various batch values

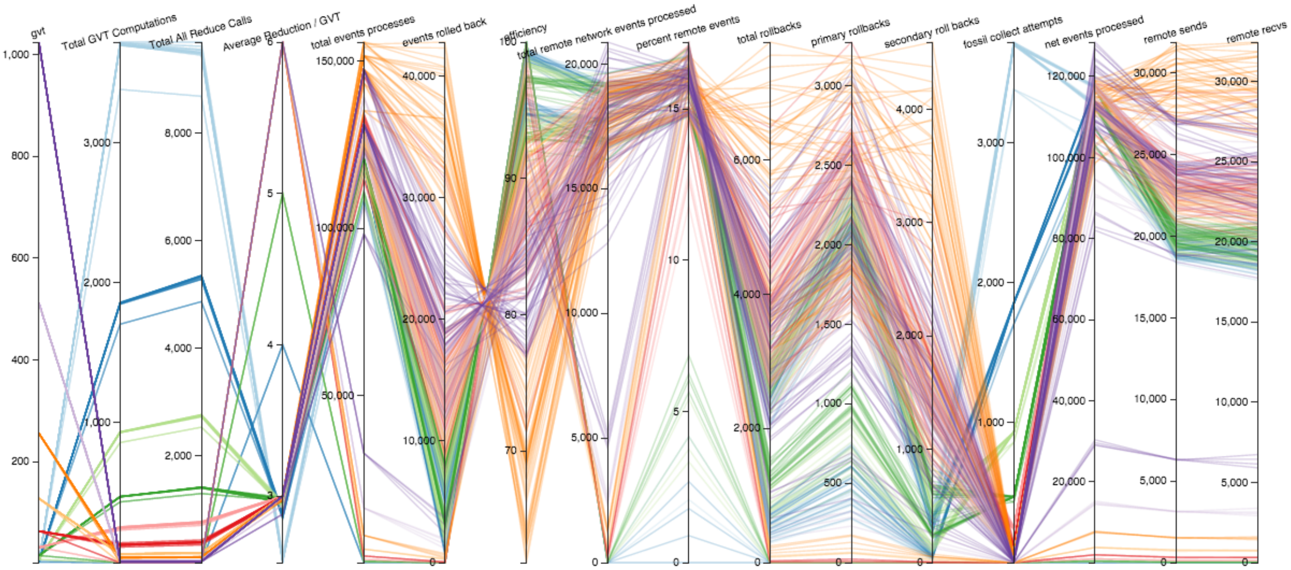


Figure 3: Parallel Coordinates graph for runs with various gvt-interval values

# 6 Visualization Tool Review

The d3.parcoords.js library used made the creation of the parallel coordinates graph very simple. Before finding this library, we tried to edit an example that used d3.js only, but it was very difficult for us to figure out how to get the various colors set for each grouping of batch or gvt_interval data points. Once we found this library, it was very simple to add in color, as well as the interactive aspects (axis shuffling and selection of a subset of the data). However, we still had difficulty with trying to change the scales of the axes. The current setup just bases it on the data, but it would be more helpful to match the axes between the two different graphs so that each metric has the same scale on both graphs.

# References

[1] D. W. Bauer Jr., C. D. Carothers, and A. Holder. Scalable time warp on blue gene supercomputers. In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, PADS '09, pages 35–44, Washington, DC, USA, 2009. IEEE Computer Society.

[2] K. Chang. Parallel coordinates. `https://syntagmatic.github.io/parallel-coordinates/`. Accessed: 2016-02-18.

[3] A. O. Holder and C. D. Carothers. Analysis of time warp on a 32,768 processor ibm blue gene/l supercomputer. In *in 2008 Proceedings European Modeling and Simulation Symposium (EMSS*, 2008.