Log2: A Cost-Aware Logging Mechanism for Performance Diagnosis

Log2 project

Software Analytics Group

Microsoft Research

May, 2015

Integration of Log2 to general logging system

Making easier integration of Log2 to an existed logging system is one of our future efforts.

Currently, there are two alternative ways to integrate Log2 to an existed logging system:

1. Replacing the original performance logging API with Log2's API
   We use static analysis technique, to automatically replace the class name from original logging system to Log2, such as:
   xLogger.Begin(xxx) → Log2.Begin(xxx);
   xLogger.End(xxx) → Log2.End(xxx);
   Note that only performance logging statements are effected.

   Additionally, Log2 needs to implement all types of performance logging APIs which the existed logging system already provided. This should be done **manually** to ensure the logical correctness.
2. **Manually** replacing the mechanism inside existed performance logging API
   This solution requires significant more efforts on incorporating Log2 into the existed logging system. While the advantage is the transparency to developers, i.e., developers just write the performance logs as usual, without any changes to their source code.

Control mechanism for utility threshold adjustment

In our paper, we adopt *Secant method* for utility threshold adjustment. This method has two important advantages: fast convergence rate (super-liner) and parameter-free. Before we figured it out, we used to try several control-based mechanism from control system domain, such as P-Control, and PI-Control for utility threshold adjustment. The major drawback of these methods are difficulties on parameter determination, i.e., if the parameters are not chosen properly, the utility threshold may oscillate and not converge at all.

It is not superise that for different software, even same software runs under different environmental setting, the "good" control parameters could varied. Figure 1 illustrates one empirical experiment, it is observed that when the parameters are set improperly, the volume of logs stored in global buffer oscillates.

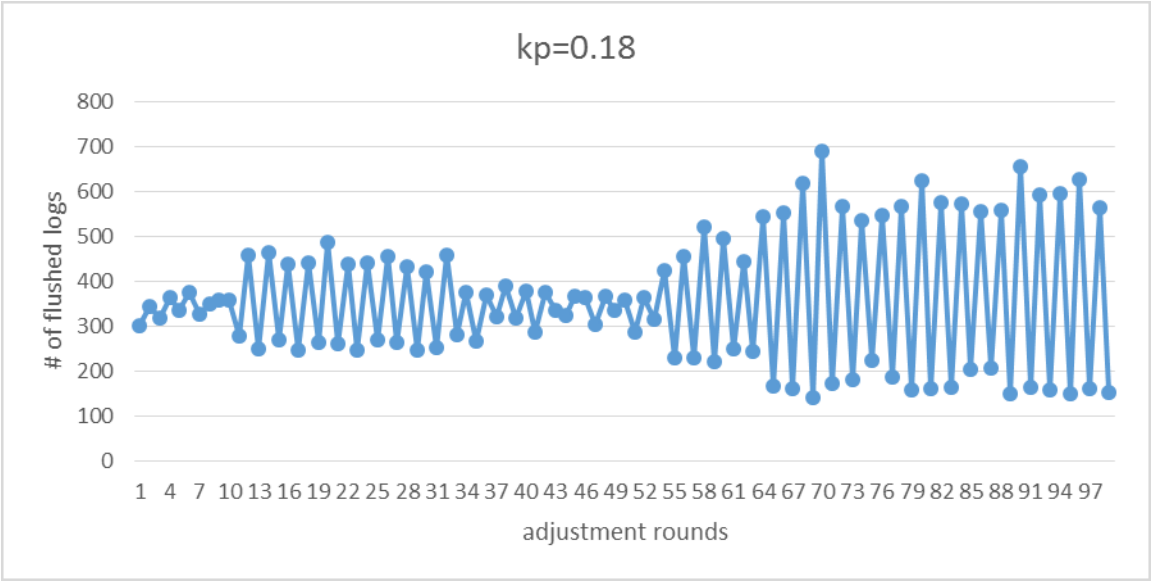

Figure 1. An illustration of global buffer size oscillation when parameters are set improperly

Advanced experiments

Design

We design two advanced experiments to evaluate *Log*^{2} for different purposes (for the whole experiment setting, please refer to Section 5 in our paper), these experiments are named ExpB and ExpC, respectively. In ExpB, a sudden change on the workload is introduced, and in ExpC, the budget is changed at a certain time. Both ExpB and ExpC are used to evaluate how well the threshold adjusting mechanism of *Log*^{2} adapts to such circumstance. Below we give illustration in detail.

Design of ExpB. ExpB is used to mimic the workloadchanging phenomenon in the real world. We increase number of users per second to increase the workload. ExpB runs for 2 hours, with 100 normal users existing in the whole duration, and 100 other normal users being added in a period of 30 minutes. In ExpB, we focus on evaluating the threshold adjusting mechanism.

Design of ExpC. In practice, it is possible that an administrator would like to tune the budget after *Log*^{2} has run for a while. ExpC runs for 2 hours; the budget size is set to a certain value during the first hour, and then is changed to double or half size (so there are two sub-experiments in ExpC) to see how effectively our threshold adjusting mechanism tackles such circumstance.

Results

Figure 2 shows the number of logs inserted into the swap buffer per each flush interval when there exist workload changes.

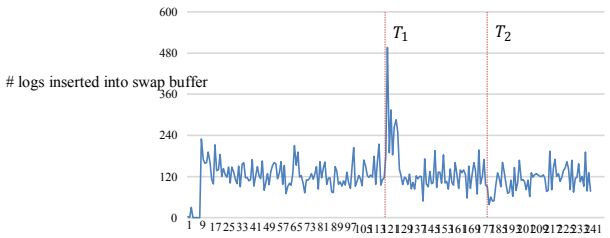

Figure 2. Dynamics of swap buffer size (changed workload)

When the workload is doubled at time *T*1, an overshoot with 3.13 times higher than average occurs. The convergence is achieved after 7 iterations. The workload turns back to normal at time *T*2, Log^2 uses 5 iterations to converge. In summary, Log^2 is agile to adapt to the sudden changes of workload.

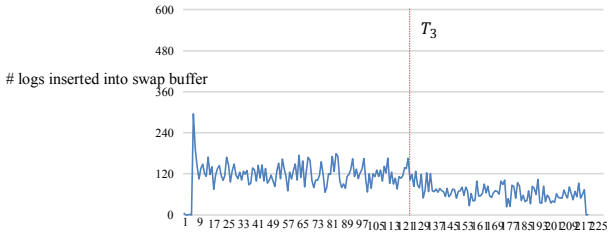

Figure 3: Dynamics of swap buffer size (budget changes)

In Figure 3, the budget size is changed to half at time *T*3, then Log^2 uses about 10 iterations to converge. Considering that the flush interval is 30 seconds, 10 iterations take only 5 minutes to finish. The memory usage in this temporary duration is negligible in practice

Additional technical details

Selection of utility scores

In current implementation of Log2, the selection of utility score is done at compile time. However, it is easy to enable configurable utility score selection at runtime, i.e., a set of pre-defined utility score functions are written in source code, and selectively triggered by a configurable flag.

Supporting mixture of various types of utility scores

Currently, one utility score function is applied to all the MCRs. It is worth thinking to loosen this restrict, i.e., assign different utility score function to different MCRs. How to conduct meaningful normalization before comparing the score together is one challenge.