1 Introduction

We would like to have statistics for profiling performance of Condor. Currently, we have very little actual information on what the various components of Condor are doing.

2 Plans

We have discussed at least two distinct plans.

2.1 Compile Time

The idea is that this implementation is cheap in terms of resources and easy to implement. But the set of possible statistics can only be defined at compile time. Running averages would be implemented in terms of ring buffers or a linked list. When clients query for statistics, accumulators run along the buffer to accumulate statistics. This would allow the daemons to compute averages or other higher level statistics. A ring buffer will use a fixed amount of memory, whereas a list can use an unbounded amount of memory; in the linked-list implementation, we will want a maximum age of events in order to keep the memory used by the list from growing without bound; this will cause statistics to be lost at particularly busy times for the daemons.

Alternatively, the burden of processing of statistics can be passed to the client. The advantage of having the statistics processing on the server is that the used in ClassAd expressions. If done on the client side, one could use the data for presentation; for example, a graphical client could do real-time display of the jobs initiated or completed in a time window.

2.2 Implementation

The implementation would look something like the following:

```
struct statAccumulator {
    int field1;
    int field2;
    time_t time_field2;
    ...
} condor_stats;

#define STATISTICS_INC_FIELD(fld) (condor_stats.fld)++
#define STATISTICS_SET_TIMED_VALUE(fld, val) \
    condor_stats.fld = val, condor_stats.time_##fld = getTime()

void statistics_update_classad(ClassAd& ad)
{
    // this is pseudocode
```

```
    foreach (fld in condor_stats) {
        ad.Update(fld, condor_stats.fld);
    }
}
```

2.3 Run Time

In a configuration file, one could define which statistics should be gathered and how often to update the statistics state. This would allow one to gather statistics generated by a ClassAd expression. An administrator could ask specifically for statistics on some Condor variable "`foo`". The advantage is that this would be dynamic and the only statistics gathering done would be that asked by the administrator. An examples of this that is already done is that some users put the core CPU temperature in their ClassAds.

3 Statistics daemon

We could implement a new DaemonCore daemon, the "statistics daemon", to offload statistics bookkeeping and calculation off the main Condor daemons.

3.1 Implementation

This would be a typical daemon. This would offload the statistics processing from the schedd, at the cost of extra communication overhead for the schedd. The Statistics daemon could publish its own ClassAds with statistics.

4 New ClassAd fields

New ClassAd fields would be defined. Erik Erlandson has identified the following statistics (see GitTrac #2006, at `https://condor-wiki.cs.wisc.edu/index.cgi/tktview?tn=2006`).

```
UpdateInterval
JobsSubmitted
JobSubmissionRate
JobsCompleted
JobCompletionRate
JobsExited
ShadowExceptions
ExitCodeXXX
JobsSubmittedCum
JobsCompletedCum
JobsExitedCum
ShadowExceptionsCum
```

ExitCodeCumXXX
WindowedStatWidth
JobsStartedCum
JobsStarted
JobStartRate
MeanTimeToStartCum
MeanRunningTimeCum
SumTimeToStartCum
SumRunningTimeCum
MeanTimeToStart
MeanRunningTime
DetectedMemory
DetectedCpus

5 Conclusion

Need input from partners and customers.