Data Mining for Effective Render Farm Management

Adam Wood-Gaines
Pixar
adamwg@pixar.com

Josh Grant
Pixar
jag@pixar.com

```
MisterD: offramp tasks active and hot   (tasks running on hosts with hot CPUs)

SQL: SELECT Task.jid AS jid,Task.tid AS tid,TaskAttempt.state AS
_TaskAttempt_state,Job.user AS _Job_user,Job.host AS _Job_host,Task.title AS
title,TaskAttempt.slots AS _TaskAttempt_PACKED_slots,TaskAttempt.statetime AS
_TaskAttempt_statetime,TaskAttempt.state AS
_TaskAttempt_state,CAST((IF(TaskAttempt.stoptime > 0 AND TaskAttempt.pickuptime > 0,
TaskAttempt.stoptime - TaskAttempt.pickuptime, IF(TaskAttempt.pickuptime > 0 and
TaskAttempt.state='active', UNIX_TIMESTAMP() - TaskAttempt.pickuptime,
UNIX_TIMESTAMP() - TaskAttempt.statetime))) as UNSIGNED) as _TaskAttempt_elapsed FROM
Task LEFT JOIN TaskAttempt ON (Task.jid=TaskAttempt.jid and Task.tid=TaskAttempt.tid
and Task.currtatid=TaskAttempt.tatid) LEFT JOIN Job ON (Task.jid=Job.jid) LEFT JOIN
Machine ON (TaskAttempt.machid!=0 and TaskAttempt.machid=Machine.machid) WHERE
((TaskAttempt.state='active') and ((Machine.name REGEXP '^(c|a|i|v)[0-9]{4}$') and
(Machine.vitaltime >= 1423701594) and (Machine.proc1temp >= 73 or Machine.proc2temp >=
73 or Machine.boardtemp >= 73))) and Job.deletetime=0 ORDER BY Task.jid,Task.tid
```

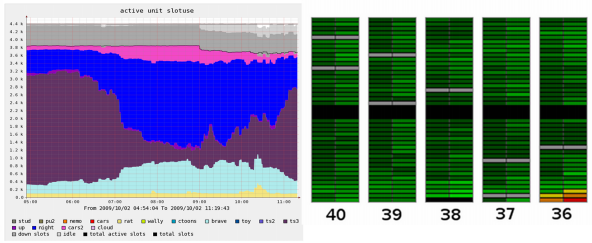

Figure 1: *A comparison of MisterD and SQL queries, a Resource Utilization Graph, and a Render Farm Heat Map.*

Abstract

In this talk we present MisterD, Pixar's render farm monitoring system. Metrics for networking, filesystem I/O, and rendering nodes, as well as the entirety of the batch rendering job queue, are stored in a relational database. A salient feature of the system is a powerful, custom query language which enables users to succinctly express complex relational queries without certain encumbrances required in a SQL syntax. We show how a number of common hardware and software failures can be efficiently detected and addressed, as well as how some uncommon pathologies have been discovered. The system has been designed to scale to very large render farms, and currently supports a farm with tens of thousands of cores. The same techniques are equally valid in smaller farms.

1 Background

The process of creating high-quality computer animated films requires vast quantities of computing resources. A failure of a single component in a sea of computing power can be difficult to detect, making the film-making process unreliable and inefficient. Widespread failure can bring a feature production to a halt. Even a well-maintained computing facility does not protect production from interruption when failures occur as software bugs or only manifest when production changes a rendering technique or its scale. Transient malfunctions can be additionally difficult to detect. Any and all of these failures become a burden on digital artists, taking away time otherwise spent realizing the director's vision. Administrators are charged with addressing these failures, so the ability to quickly detect and respond to them are of paramount importance.

2 Implementation

At the center of MisterD is a relational database that stores the status of all rendering nodes and their processes. It is with this knowledge base that both administrators and automated processes can identify problems and take corrective measures so that production can receive maximum benefit from the computing resources.

A daemon on each compute node collects metrics such as memory and disk use, file system I/O, and running process statistics. The daemon will update the database only if there is a significant change in these values, greatly reducing the database server load. A custom connection pool is used to scalably manage the thousands of persistent database connections between the nodes and the database server.

Job scheduling is performed by a horizontally scalable collection of servers, which schedule millions of tasks per day. They utilize the same database for their persistent store, so the status of any job is known at any point in time and can be related with other system metrics when queried. These metrics also inform scheduling decisions.

The database is further populated by a process that categorizes every failed render by inspecting the render logs and node state on process exit. This facilitates the automation of responses such as retrying all renders that failed due to a read-only drive.

Additionally, statistics for completed renders are archived, which is used for estimating CPU and memory when scheduling future batch jobs and for budgeting of computing resources for future films.

3 Tools

The collected data drives a large number of tools, including a command line tool for job wrangling and node administration, a render farm heat map visualization, the automation of offlining problematic nodes, a paging system that alerts admins of component failure, and an extensive graphing system that charts network, filesystem, and compute resource utilization by various organizational groupings.

Central to each of these tools is the use of the custom query language for creating their search clauses. Search clauses are eventually translated to SQL, though they are much easier to read and to write than their SQL counterparts. The query language has a minimalist ethos, eliminating punctuation where possible and supporting readable abbreviations for units of time and space. The translation process infers the required table joins based on the terms in the search clause. Lastly, aliases permit simple terms to represent arbitrarily complex queries.

SIGGRAPH 2015 Talks, August 09 – 13, 2015, Los Angeles, CA.
ACM 978-1-4503-3636-9/15/08.
http://dx.doi.org/10.1145/2775280.2792538