October 2006 Volume II Issue 4

ORAtips Journal

ORAtips On Database

Subgrid Load Balancing for Utility Computing

By Todd Morley

Editor's Note: *Grid computing, what's all the buzz? We asked Todd Morley to give us his view on demystifying the process for load balancing across subgrids. The result is an informative piece that discusses the challenges of a scheduling system that manages a grid over multiple servers, each running multiple Xen virtual machines and the scripts to make it a seamless exercise. Todd's naïve PL/SQL and test seed data scripts are enough for two database articles, so we posted them to our Website www.ORAtips.com Document Library under Database – Grid Computing.*

Introduction

Virtual-machine technologies create new opportunities and challenges for managing utility-computing grids. When the grid consists of several racks of servers, each serviced by a dedicated network switch, one challenge is balancing loads across the racks (subgrids). This article describes an Oracle®-based approach to balancing virtual-machine loads across subgrids. The author developed the approach as part of a scheduling system that manages a grid of over 400 physical servers, each running multiple Xen virtual machines. The virtual machines can start and end at arbitrary points in time, and can support loads of arbitrary magnitudes.

Assumptions

This article makes the following assumptions:

1. A subgrid is a rack of several physical servers, serviced by a dedicated network switch.

2. A grid consists of several subgrids.

3. Each physical server can host several virtual machines.

4. Each subgrid is assigned a schedulable percentage of its servers' total physical RAM.

5. A load on the grid, or instance, has the following characteristics:
   a. start datetime
   b. end datetime
   c. nominal RAM requirement
   d. instance type.

6. Each subgrid is allocated a fixed range of IP addresses and hostnames for each instance type.

7. Instances are allocated to support events (such as training classes, product demonstrations, and system tests). For our purposes, an event is a collection of instances. Each instance's start time is the event's start time, less an instance-specific lead time ("pre-gap"). Each instance's end time is the event's end time, plus an instance-specific lag time ("post-gap"). (The lead and lag times are presumably sufficient to set up and clear software and data in whatever disk space is allocated to the instance—a technical issue that this article glosses over.)

8. The scheduling algorithm is responsible for assigning instances to subgrids.

9. Requests to allocate instances appear in a single queue, and are processed in serial.

The scheduling algorithm implements a mini-max heuristic to balance the subgrids' loads.

This article's discussion suggests how the architecture can be extended to

- balance loads within subgrids
- use lists (rather than ranges) of instance numbers for instances types
- handle concurrent requests.

For purposes of illustration, this article assumes that there are three instance types, labeled "ab", "cd", and "ef". (This article makes no assumptions about instance-type semantics. In the author's context, there are different instance types for production, test, and development instances, among others.)

Schema

The load-balancing architecture requires two database tables. One table represents subgrids:

```
create table subgrid(
id integer,
name varchar2(100),
rack varchar2(100),
physical_ram_capacity_gb number,
max_percent_ram_to_schedule number,
ab_lower_limit integer,
ab_upper_limit integer,
cd_lower_limit integer,
cd_upper_limit integer,
ef_lower_limit integer,
ef_upper_limit integer,
online_yn varchar2(1)
);
```

Most of the columns in the subgrid table should be self-explanatory. The lower- and upper-limit columns define mutually exclusive ranges of instance numbers allocated to each instance type, for a given subgrid. An instance's number and type combine to produce a hostname. For example, at Foo Incorporated, the instance number 101 and instance type ab might combine to produce the hostname "ab101.foo.com", which would be assigned to the instance's virtual machine. Presumably the network is configured so the network switch on the subgrid having ab_lower_limit <= 101 and 101 <= ab_upper_limit services that hostname.

Here is the definition of the table representing scheduled instances:

```
create table subgrid_load(
event_id integer,
event_utc_start_date date,
event_utc_end_date date,
load_utc_start_date date,
load_utc_end_date date,
ram_requirement_in_gb number,
instance_type varchar2(2),
subgrid_id integer,
instance_number integer,
instance_name varchar2(50)
);
```

Note that the subgrid_load table only captures an instance's lead and lag times implicitly, by subtracting (resp. adding) each to the event start and end dates, and storing the results as load start and end dates. How these tables are indexed will vary with the implementation of the algorithm described in the sequel, so we gloss over the issue here.

Algorithm

The scheduling algorithm implements a minimax heuristic to balance the subgrids' loads. When presented with an instance to assign to a subgrid, the algorithm chooses a subgrid that minimizes the maximum difference between the least- and most-loaded subgrids, over the input instance's lifespan. This greedy approach is relatively simple and efficient.

Here is pseudocode for the minimax algorithm:

```
01 for each subgrid {
02 if {
03  the subgrid is offline or
04  the subgrid has no schedulable RAM or
05  the subgrid has no schedulable instances of
    the required type
06 }
07 then {
08  continue
09 }
10 maxLoad = 0
11 for each load start time within the input load's
   lifespan {
12  if {
13   the total load at the load start time >
     maxLoad
14  }
15  then {
16   maxLoad = the load start time
17  }
18 }
19 normalizedMaxLoad =
20  maxLoad / the current subgrid's max sched-
    ulable RAM
21 if {
22  (no optimal subgrid has been found or
23  normalizedMaxLoad < optimalNormalized-
    MaxLoad) and
24  the current subgrid has an unscheduled
    instance and
25  the current subgrid has sufficient unscheduled
    RAM
26 }
27 then {
28  optimalNormalizedMaxLoad = normalized-
    MaxLoad
29  optimalSubgrid = current subgrid
30 }
31 }
32 choose an instance number that is unscheduled
   over the input lifespan on optimalSubgrid, for
   the input instance type
33 return the chosen instance number (or the
   related instance name)
```

The algorithm has two loops: an outer loop starting at line 01, and an inner loop starting at line 11. The outer loop iterates through the subgrids, looking for an optimal subgrid—that is, a subgrid that satisfies the minimax heuristic. The if statement in lines 02-09 filters out subgrids that are offline or lack resources (allocated or not). Lines 10-20 identify a subgrid's maximal load, over the lifespan of the instance to be assigned. Lines 21-30 set the optimal subgrid to the current subgrid, if the current subgrid is optimal and has sufficient available resources.

The algorithm has four main subtleties:

1. Note the distinction between schedulable resources in lines 4-5 and unscheduled resources in lines 24-25. A subgrid has schedulable resources (instance numbers and RAM) in the abstract, independent of any given instance. A subgrid has unscheduled resources if the resources are available during a specific instance's lifespan.

2. The algorithm determines the current subgrid's maximal load, within the input instance's lifes-

pan, by checking the load at only a subset of the times at which the load can change. To understand why it suffices to check just the subset, consider that a subgrid's load over an input instance's lifespan is a step function. Its set of distinct values can be exhausted by examining the subgrid's load at four kinds of points in time:

- the input instance's start date
- the input instance's end date
- the start dates of previously scheduled instances (falling in the input lifespan)
- the end dates of previously scheduled instances (falling in the input lifespan).

An elementary case analysis demonstrates that the load at the input end date and previously scheduled end dates are weakly dominated by the load at the input start date and previously scheduled start dates. That is, some load in the latter set will at least equal all loads in the former set. So it suffices to examine the load for the latter set. This is the set identified at line 11 of the pseudocode.

3. The pseudocode does not specify (at line 32) how to choose an instance number that is unscheduled over the input load's lifespan, for the optimal subgrid and the input instance type. To make this choice, one must identify all scheduled instances of the input type that overlap the input instance's lifespan, on the optimal subgrid. These instance numbers are unavailable for assignment to the input instance. One can then choose (say) the lowest unscheduled instance number in the optimal subgrid's instance-number range for the input instance type.

   To identify previously scheduled instances of the input type that overlap the input lifespan on the optimal subgrid, another case analysis suffices. Let SI and EI represent the start and end datetimes of the input instance, and let SO and EO represent the start and end datetimes of some previously scheduled instance. There are six possible orderings:

   SI EI SO EO
   SI SO EI EO
   SI SO EO EI
   SO EO SI EI
   SO SI EO EI
   SO SI EI EO

   Four of these six cases represent overlapping lifespans. In two of these cases, the other instance's start date (in bold) falls in the input instance's lifespan (in italics); in the other two, the reverse is true. So, to query for instance numbers (and names) that are in use during an input instance, the following where-clause conditions suffice:

```
load_utc_start_date between
(eventUtcStartDateIn - preGapInDaysIn)
and
(eventUtcEndDateIn + postGapInDaysIn)
or
(eventUtcStartDateIn - preGapInDaysIn)
between
load_utc_start_date and
load_utc_end_date)
```

   This article's PL/SQL implementation of the algorithm uses these conditions.

4. Finally, it is important to recognize that the pseudocode is neither efficient nor inefficient. It merely communicates the logical structure of the minimax heuristic. The heuristic has many expressions, each more or less efficient, in many programming languages. One must focus on efficiency in a particular implementation of the heuristic.

Tests

The following tables store a set of test cases for the algorithm:

```
create table subgrid_load_test(
id integer,
description varchar(100)
);

create table subgrid_load_test_subgrid_up(
test_id integer,
subgrid_id integer
);

create table subgrid_load_test_instance(
test_id integer,
event_id integer,
event_utc_start_date date,
event_utc_end_date date,
ram_requirement_in_gb number,
instance_type varchar2(2)
);
```

In the white paper version of ths article found at http://www.oratips.com/AccessDocumentCategories.asp?menuID=24, Appendix 2 contains code that seeds 15 tests into these tables. The tests probe a variety of possible logical errors in an implementation of the algorithm. Table 1 gives brief descriptions of the tests. (Each of the tests starts with no previously scheduled instances.)

The subgrid_load table only captures an instance's lead and lag times implicitly...

Discussion

Four aspects of the approach described in Table 1 deserve further discussion.

Performance and scalability. With elementary indexing on the subgrid_load table, in a modest development environment, and no other optimizations, the algorithm executed in at most 0.06 seconds of real time (measured using dbms_utility.get_time, so these measurements are upper bounds on actual execution times) in all of the test cases. The execution times were 0.00 or 0.01 seconds in most cases.

Clearly, however, the test cases are too small to probe performance under realistic load conditions. The author's team schedules instances up to three months in advance. The typical instance lasts one week, and may require 10-100% of a physical server's schedulable RAM, with a typical instance requiring perhaps 25%. For a grid of (say) 500 physical servers, this works out to a total load of up to 2,000 instances operating at any given time, and up to 24,000 instances scheduled at any given time.

The author's team is currently migrating its grid to a subgrid architecture, bringing one subgrid online every week or so.

Balancing loads within subgrids. It is desirable to assign instances to subgrids several weeks in advance, so that their instance numbers (and hence the hostnames of their virtual machines) are known. It is neither necessary nor desirable to assign instances to fixed physical servers with the same lead time. One reason is, a physical server may have an unscheduled downtime as an instance's start time approaches. Delaying assignment of an instance to a physical server until the instance's start time arrives minimizes the impact of unscheduled downtimes.

Having said that, it is possible to adapt this article's approach to balance loads within a subgrid. In the adaptation, the physical server takes the role of a subgrid. So, a physical-server table replaces the subgrid table, and the physical-server ID replaces the subgrid ID. A virtual machine on the physical server is then started and bound to the instance. This article omits details.

Handling arbitrary instance types. The architecture described in this article can be made more robust by adding to it the following table:

```
create table subgrid_instances(
subgrid_id integer,
instance_type varchar2(2),
instance_number integer
);
```

Each row in subgrid_instance would represent an instance name (the combination of type and number) assigned to a subgrid. (It is possible further to normalize the architecture by introducing a separate instance_type table, but one questions whether a table with so few rows would justify the extra join it would require at runtime.) The code in Appendix 1 as part of the white paper version of this article located at http://www.oratips.com/AccessDocumentCategories.asp?menuID=24 would be altered to query subgrid_instances in the code that realizes line 32 of the pseudocode (the choice of minimal available instance number), in particular. The lower- and upper-limit columns would be dropped from the subgrid table, and there would be no requirement that a subgrid support a range of instance numbers. A subgrid could support an arbitrary set of instance numbers, and a grid could support an arbitrary, extensible set of instance types, at the cost of an additional join in some queries.

Handing concurrent requests. This article's ninth assumption essentially limits instance-scheduling requests to a single source. In practice, such requests may come from several sources—a Web interface for ad-hoc requests, as well as an automated scheduling system, for example. The author expects to investigate using Oracle Advanced Queuing and several other approaches to the concurrency problem in the near future, and hopes to report his results along with performance-related results in a subsequent article.

Conclusion

This paper has described an Oracle-based minimax heuristic for balancing loads across subgrids in a grid running Xen virtual machines. The described naive code runs quite efficiently for all of the tests, but scalability may require modifications to the naive implementation. Multiple scheduling-request sources may also require modifications. Another interesting question is how well the minimax heuristic utilizes grid resources, compared to a batch-scheduling algorithm that attacks the load-balancing problem from a more global perspective.

Todd Morley, *Oracle Corporation* - Todd has been with Oracle since 1995. He is currently an architect in Oracle's Global IT organization. Previously he was the sole architect of Oracle Approvals Management (AME) for the first five years of that product's existence. His AME architecture earned two patents (one granted, the second pending). Todd did his doctoral work (ABD) in industrial engineering at Stanford. Todd may be contacted at Todd.Morley@ERPtips.com.

This article was originally published by Klee Associates, Inc., publishers of JDEtips and SAPtips. For training, consulting, and articles on JD Edwards or SAP, please visit our websites: www.JDEtips.com and www.SAPtips.com.