Contract Renegotiation

A whitepaper discussion of rescheduling and contracting
in the GrADS environment.

Otto Sievert
University of California, San Diego
13 January, 2001

1 Introduction

Users of computational grid software have expectations about the performance of their applications. However, applications that execute on computational grids can be succeptible to large changes in performance due to the inherently dynamic nature of grid resources. Steerable applications can change their resource requirements during execution, also causing performance variation. One way to address these kinds of run-time variations, and thereby meet users expectations, is to dynamically *reschedule* the application by evaluating performance during execution and reacting appropriately to meet users' expectations.

To address the many technical aspects of rescheduling, the concept of a *performance contract* has been developed. A performance contract contains information about an application and the resources on which it executes. It assists in determining when application performance is unacceptable, and assists in determining the cause of the poor performance. In this context, rescheduling can be thought of as a form of *contract renegotiation*.

As a concept, contracts provide exactly the features needed to meet users' needs. However, contract details are not well understood or documented. This document attempts to define some of the characteristics of contracts that are necessary for rescheduling in a GrADS-like environment. We will first define some terms and describe this environment, then give a rescheduling analysis by approaching the topic from a fundamental viewpoint, giving treatment to the symptoms of a contract violation, the causes of such, and possible contract renegotiation techniques.

2 Definitions

For the simple reason that it is difficult to address any topic without a concrete language in which to express ideas, we present this glossary of terms before our main discussion. Undoubtedly not everyone will agree with these definitions, but they provide a base vocabulary for the specific discussions that follow.

resource *n* a physical device which can be used to perform work. This device may possibly be reserved, and may possibly be allocated, but is most commonly shared. Examples: Cray T90, Sun workstation, 10Mbit ethernet.

virtual machine *n* **1** a concrete collection of resources selected specifically for a particular application, and the topological relationship among these resources. For example, the virtual machine of a master/worker application might be the machines `dralion.ucsd.edu`, `soleil.ucsd.edu`, `magie.ucsd.edu`, and `mystere.ucsd.edu`, where `soleil` is the master and the others are workers. **2** an abstract reference to such a collection of resources.

application performance *n* any of a number of metrics describing important application execution characteristics. Examples: execution time (this is the typical meaning), execution rate (flop rate), frame rate (for interactive graphical applications).

dynamic resource *n* a resource whose availability or performance changes over time.

resource reservation *n* the guarantee of partial or full allocation of a resource for a specific period of time.

schedule *v* to determine an application's execution resources, work allocation, and other run-time configuration information.

schedule *n* a concrete description of an application's execution resources, work allocation, and other run-time configuration parameters.

scheduler *n* an entity that schedules. Typically schedulers do not actuate the application, they instead determine the time and place of execution and configure salient application run-time parameters.

reschedule *v* to re-examine and possibly alter an application's execution resources, work allocation, and other run-time configuration parameters, creating a new or altered schedule. Most commonly this action occurs after execution has begun, but could occur at any time after an initial schedule is created. The act of rescheduling can be thought of as scheduling with initial conditions describing the existing schedule and execution environment.

rescheduler *n* an entity that performs rescheduling

performance contract *n* a binding set of resource reservations, schedules, and absolute performance information that guarantee a specific application performance and enable measurement of that performance. Also "contract".

contract negotiation *v* the act of creating an application performance contract.

contract negotiator *n* entity that creates a performance contract.

contract renegotiation *v* see **reschedule**.

contract renegotiator *n* see **rescheduler**.

contract violation *n* event that occurs upon detection that an application's contracted performance (a) has not been, (b) is not being, or (c) will not be achieved.

police of G *n* the entity that monitors both application execution behavior and resource behavior. It has the power to declare a contract violation. This is a neutral third-party citizen of the grid whose job includes resolving disputes between resources and applications. In our case, the police of G provides information to help determine why a performance contract violation occurred. Also "monitor".

application performance model *n* a parametric model that predicts absolute or relative application performance.

resource selector *n* the part of the scheduler that selects grid resources appropriate for a particular application based on application characteristics contained in the AART.

AART *n* Application Abstract Resource Topology. A collection of descriptive and parametric resource characteristics desired by an application. The AART parameters describe application needs independent of any given run-time data. For example, the AART contains the kind of resource topology (e.g., machines are arranged in a two-dimensional mesh) without mention of the exact size of such a mesh (which is highly data and resource dependent).

3 Overview

We explore rescheduling in a rich program preparation and execution environment that affords many luxuries uncommon in today's typical environment, including access to application innards (at a source and/or object code level) and the ability to monitor application execution. This elaborate system is primarily striving for:

- ease of use for disciplinary scientists running grid applications

- application performance
- generic use (not tied to any one specific application)

We assume a specific scenario in which a GrADS-enabled application is executing on a set of resources. These resources are and have been monitored by a performance monitoring system such as the Network Weather Service (NWS). The application is instrumented with probes that also track salient execution details; the Autopilot system provides such capability. Prior to execution, the resources in use signed a performance contract guaranteeing a particular execution time. The details of this contract are glossed over now, as the exact nature of the contract is one of the items this work will address.

Within this framework, our scenario further requires that, for some reason, the police of G declares a breach of contract. The rescheduler, or contract renegotiator, is called upon to remedy the situation. Using only the information supplied by the application and environment monitoring systems and the contract paperwork itself, the rescheduler must determine an appropriate course of action.

With our stage thus set, we proceed to examine the situation from every angle, looking for a thorough understanding of the circumstances and our possible responses. We begin by discussing the possible reactions we have available, then explore the fundamental reasons for poor performance, and finally examine the symptoms that these causes exhibit.

4 Reactions

We will temporarily avoid discussing the symptoms of poor performance. Instead, we start by addressing our possible reactions to any contract violation. They are:

- We can always decide to do nothing, continuing execution at the current rate.
- We can stop the current execution, and perform a full restart including a full scheduling / resource selection. [aside: how do we not make the same mistake twice?]
- Repurpose the existing resources. We can compute a better distribution of the remaining work across the existing resource set.
- Remove resource(s). The abandoned work must be allocated to the remaining resources.
- Replace resource(s). Suitable replacements must be selected, and the abandoned work, data, processes, etc., migrated to the new resource(s).
- Augment resource set. Suitable additional resources can be selected, and added to the virtual machine. Work must be allocated to these new machines from the existing workload. Some data migration may be necessary.
- We can adjust application intrinsic execution parameters (such as graphics frame rate or mesh resolution) to increase performance.
- We can combine one or more of these actions.

5 Causes

Next we define a taxonomy of reasons why application performance could be poor. [aside: is poor performance is the only reason why a contract will be violated?].

1. One resource is not computing or communicating to the promised degree as measured by the resource monitors. Perhaps the load has changed since execution began, or the monitor data is simply inaccurate or noisy.
2. The application is not executing as promised. Perhaps the number of loop iterations peformed is more than predicted, or the amount of work has increased.

3. The resource measurements given to the performance model do not accurately conform to the performance model assumptions. For example, the application in question does not communicate in the same amount or at the same frequency as the communication resource measurements. [ref. “Adaptive Performance prediction for Distributed Data-Intensive Applications”, Marcio Faerman et al., Proceedings of the ACM/IEEE SC99 Conference on High Performance Networking and Computing]

4. The application performance model is inaccurate: even when given accurate values the model does not accurately predict this application’s performance. Perhaps the model was automatically generated, is of too general a form, or does not account for significant performance-effecting parameters.

5. A combination of any number of the above possibilities. For example, more than one resource is bad, or three resources are bad and the application is performing twice the amount of work it promised.

6 Symptoms

Finally we discuss the symptoms of poor performance. Our symptoms are limited only by our ability to observe the system. We can observe the following phenomena:

- the configuration of the application and the virtual machine before and during execution.
- the cpu load of each computing resource in the virtual machine.
- the memory consumption of each computing resource in the virtual machine.
- the bandwidth and latency of each communication link in the virtual machine (or representatives of such links).
- the wall clock time.
- application intrinsic performance measures (not all of which may be available for all applications):
  - communication synchronization barrier wait time
  - communication amount and frequency
  - amount of computation (via hardware counters, knowledge of flops per loop iteration, etc.)
  - computation rate
  - amount and frequency of disk I/O
  - per-application special metrics like loop iterations or frame rate or mesh resolution

Our primary job, now, is to deterministically match symptoms with causes, and then to apply the appropriate corrective action. Sounds simple, right? These symptoms will be combined using heuristics to make an educated guess as to the cause of a performance contract violation. For example, if barrier wait times are large on all resources but one, and this resource is heavily loaded, then it is likely that the computation rate of this resource is the performance inhibitor.

In any case, we need a place to gather all this information that will be used in performance triage, and the contract is the natural choice.

7 Contract

In this scenario, to properly gather the information necessary for contract violation analysis and rescheduling, our contract comprises the following:

- the exact definition of the term “performance” (e.g., execution time).
- the current schedule:
  - the virtual machine in use.

  - predicted (negotiated) performance of the virtual machine (environment intrinsic characteristics).
  - actual values of these characteristics during execution.
  - current work allocation.
  - other application execution parameters.

- a description of the salient application intrinsic characteristics.
- predicted (negotiated) values for these application intrinsic characteristics.
- actual values of these characteristics during execution.
- the ability to query current values for both environment intrinsic and application intrinsic metrics.
- the ability to predict future values for both environment intrinsic and application intrinsic metrics.
- an absolute performance model that calculates current performance and predicts final performance, based on current and predicted environment intrinsic and application intrinsic metrics.

This contract must be specified in a standard format that does not preclude creation by an application developer, a library developer, a user, or the compiler.

The rescheduler will use the contract information to predict the costs and benefits associated with (a) continuing execution with no changes, (b) completely halting the application and scheduling from the beginning (a full reschedule), and (c) a partial reschedule that uses one or more of the reactions discussed above, which was arrived at by a causal analysis based on the execution symptoms. The action taken will be the best of these three options.

Another rescheduling option that does not involve the police of G would require that the application have defined discrete checkpoint/restart/reschedule waypoints prior to execution. After each of these *units of progress*, or at the last achieved waypoint if we don't reach the next progress waypoint in a certain amount of time, we would perform a rescheduling as above and proceed accordingly.