Prato: databases on demand

Soila Pertet, Priya Narasimhan
Carnegie Mellon University
Pittsburgh, PA 15213
{spertet,priyan}@ece.cmu.edu

John Wilkes, Jay J. Wylie
HP Laboratories
Palo Alto, CA 94304
{john.wilkes,jay.wylie}@hp.com

Database configuration can be a daunting task as database administrators are often presented with a myriad of configuration options that are difficult to sift through. Prato, a project at HP Labs, is a prototype of a self-managing DBMS service provider that eases this burden by using economic incentives to guide automated DBMS setup and management. Prato offers customers private, virtual, DBMS appliances that can each be sized up to several hundred nodes, and made available on demand, in just a few minutes.

1 Introduction

Configuring a database is much easier than it used to be, but it is still a chore that many of us would rather not perform, and one that continues to be quite error-prone [4]. The investment in hardware and people necessary to make a large, powerful database available is prohibitive for short-term tasks. As a result, applications that might use a database do without; queries take much longer than they need to; and people waste time learning how to make their database fast, rather than getting on with what they really wanted to do.

Prato is a prototype of a self-managing DBMS service provider that solves these problems by offering customers private, virtual, DBMS appliances that can be sized up to several hundred nodes, and made available on demand, in just a few minutes. Prato's research goals are to learn how best to:

1. *capture customer needs*, without dictating the way those needs are addressed (in our case, what resources to allocate to the database, and how important it is that it stays available);
2. *automatically translate such customer needs into implementation choices* (in our case, what kind of DBMS options to set, and how much to invest in different failure-tolerance mechanisms); and
3. *completely automating the management of the Prato service provider*: lights-out self-management is the end goal, even in the face of conflicting user requirements, and failures.

The initial Prato prototype focuses on making the DBMS resilient to a wide range of failures. The prototype uses economic rewards and penalties to drive automated decision making. In particular, Prato uses penalty costs in its service contracts to determine which of several failure-tolerance mechanisms to employ for a particular virtual DBMS appliance. Research on the failure diagnosis and recovery aspects of Prato is being done in collaboration with CMU.

2 Prato system architecture

The system architecture for Prato (Figure 1) consists of a *service-delivery* side (shown on the right), comprising the DBMS that hosts the customer's database, the machines on which runs, and file storage; and the *service manager*, which is responsible for running the service and controlling the other parts.

The novel part of Prato is the service manager. It is in charge of controlling the execution of all the externally-visible Prato functions. These fall into three main categories:

1. *create/relinquish a DBM*S (invoked by the customer in their Manager role).
2. *upload/download data*, which also performs schema and table instantiation (invoked by the database-administrator role of the customer).
3. *query execution*, provided via an ODBC feed (performed by the customer in their Analyst role). Note that this runs at full speed, using the native DBMS access methods, rather than going via the Service Manager.

Prato uses the WX2 DBMS from Kognitio [2], a high-performance and scalable DBMS for business analytics. Other back-ends for Prato are in the works.

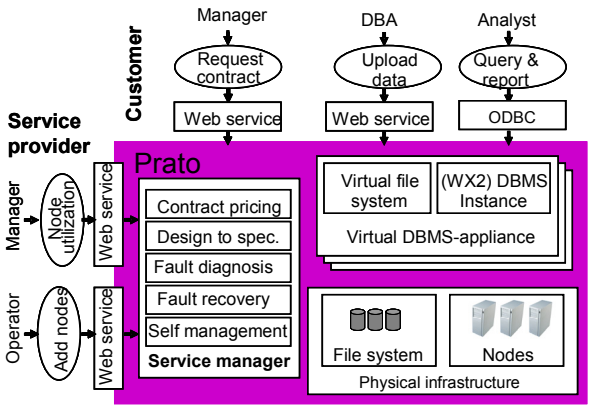

Figure 1: Prato system architecture.

Fourth International Conference on Autonomic Computing (ICAC'07)
0-7695-2779-5/07 $20.00 © 2007 IEEE

Business-manager and operator roles are defined for the service provider, but our goal is to make these as little needed as possible.

The service manager is implemented using the Enigmatec EMS system [1]. EMS provides location independent workflow management and a convenient engine for implementing policies. The Service Manager is responsible for designing virtual DBMS appliances that meet client contract requests, pricing such contracts, allocating physical resources for such contracts, instantiating a specific virtual DBMS appliance on allocated resources for accepted contracts, diagnosing failures, and planning recoveries from failures.

3 Failure-tolerance

The current implementation of Prato automates the task of creating and relinquishing a DBMS, and runs on a test bed with over 200 HP DL360 nodes, which have 400 processors, 0.8TB of RAM, and 7.2TB of disk capacity. The first Prato prototype focuses on making the DBMS resilient to a wide range of failures, a key aspect of being completely self-managing.

Customers submit their requests for a DBMS via a web-based interface by specifying how long they would like to rent the DBMS, and the size of their database in terms of the memory and disk space it requires. In addition, customers detail the penalties that the service provider will incur in the event of service unavailability or data-loss.

For example, a customer might specify that the service provider will incur a penalty of $10,000 for every hour that the service is unavailable. This penalty-based specification allows us to evaluate the merits of different failure-tolerance mechanisms in terms that are meaningful to the customer, rather than the system administrator.

Table 1 lists the failure-tolerance mechanisms that mitigate the loss of data in the event of a DBMS crash or hardware failure. These mechanisms range from solutions with low outlays but long recovery latencies, e.g., reloading the customer's data from a local cached copy, to solutions with high outlays but short recovery latencies, e.g., hot standby DBMS.

Solutions with low outlays are preferred when the penalties are low; similarly, solutions with high outlays are preferred when the penalties are high. The current prototype supports RAID 5 and mirroring. Future versions will support the full range of failure-tolerance mechanisms.

Our first prototype uses a static mapping from penalties to mechanisms, based on work done by Keeton et al. [3]. We plan to change this.

Failure-tolerance mechanism	Protects against
reload from local copy	loss of uploaded data
periodic database snapshots	loss of query updates
RAID 5; mirrored disk	disk failures
dedicated/hot spare nodes	node failures
cold/hot standby DBMS	node failures

Table 1: Failure-tolerance mechanisms.

One of the challenges we face is structuring the failure diagnosis and recovery services to minimize service outages in highly-dynamic, large-scale distributed systems like Prato, where resources are assigned to different resource managers during a contract's lifetime. Conflicts can arise when multiple resource managers simultaneously detect a resource failure and attempt to resolve it. Policies which clearly outline responsibilities for failure diagnosis and recovery are critical for orchestrating failure recovery.

Another challenge is adapting our static algorithms to cope with uncertainty in component failure rates, as failure rates in the field can differ widely from the manufacturer's datasheet [5]. We are interested in formulating policies to help service providers maximize their profits, even in the face of uncertainty.

4 Summary

Prato is a service provider that eases the burden of DBMS management by using economic rewards and penalties to guide automated decision making. By deploying Prato, we expect to gain insights into how to design and build lights-out self-managing information services, and to identify follow on research projects that must be solved before on-demand service provision can become common place.

5 References

[1] Enigmatec website. http://www.enigmatec.net/

[2] Kognitio website. http://www.kognitio.com/

[3] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes. *Designing for disasters*. Proc. 3rd USENIX Conference on File and Storage Technologies (FAST'04), pp. 59–62, 2004.

[4] F. Oliveira, K. Nagaraja, R. Bachwani, R. Bianchini, R. P. Martin, and T. D. Nguyen. *Understanding and Validating Database System Administration*. Proc. USENIX, pp 213–228, 2006.

[5] B. Schroeder, and G. Gibson. *Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?* Proc. 5th USENIX Conference on File and Storage Technologies (FAST'07), pp. 1–16, 2007.