Distributing and Managing Data on Desktop Grids with BitDew

Gilles Fedak, Haiwu He, Franck Cappello
INRIA Saclay, Grand-Large, Orsay, F-91893
LRI, Univ Paris-Sud, CNRS, Orsay, F-91405
{fedak,haiwuhe,fci}@lri.fr

ABSTRACT

Desktop Grids use the computing, network and storage resources from idle desktop PC's distributed over multiple-LAN's or the Internet to compute a large variety of resource-demanding distributed applications. While these applications need to access, compute, store and circulate large volumes of data, little attention has been paid to data management in such large-scale, dynamic, heterogeneous, volatile and highly distributed Grids.

To address this problem, we propose the BitDew framework, a programmable environment for automatic and transparent data management on computational Desktop Grids. BitDew relies on a specific set of meta-data to drive key data management operations, namely life cycle, distribution, placement, replication and fault-tolerance with a high level of abstraction. The Bitdew runtime environment is a flexible distributed service architecture that integrates modular P2P components such as DHT's for a distributed data catalog and collaborative transport protocols for data distribution.

Categories and Subject Descriptors

H.3.3 [**Information Systems Applications**]: Information Storage and Retrieval—*Information Storage and Retrieval*

General Terms

Design, Experimentation, Performance, Management

1. INTRODUCTION

Enabling Data Grids is one of the fundamental efforts of the computational science community as emphasized by projects such as EGEE and PPDG. This effort is pushed by the new requirements of e-Science. That is, large communities of researchers collaborate to extract knowledge and information from huge amounts of scientific data. This has lead to the emergence of a new class of applications, called *data-intensive* applications which require secure and coordinated access to large datasets, wide-area transfers and broad distribution of TeraBytes of data while keeping track of multiple data replicas. The Data Grid aims at providing such an infrastructure and services to enable data-intensive applications.

UPGRADE-CN'08, June 23, 2008, Boston, Massachusetts, USA.
ACM 978-1-60558-155-2/08/06.

Our project, BitDew[1], targets a specific class of Grids called Desktop Grids. Desktop Grids use computing, network and storage resources of idle desktop PCs distributed over multiple LANs or the Internet. Today, this type of computing platform forms one of the the largest distributed computing systems, and currently provides scientists with tens of TeraFLOPS from hundreds of thousands of hosts. Despite the attractiveness of this platform, little work has been done to support data-intensive applications in this context of massively distributed, volatile, heterogeneous, and network-limited resources. Most Desktop Grid systems, like BOINC [1] or XtremWeb [3] rely on a centralized architecture for indexing and distributing the data, and thus potentially face issues with scalability and fault tolerance.

However, we believe that the basic blocks for building BitDew can be found in P2P systems. Researchers of DHT's (Distributed Hash Tables) and collaborative data distribution [2, 4], storage over volatile resources [6] and wide-area network storage [5] offer various tools that could be of interest for Data Grids. To build Data Grids from and to utilize them effectively, one needs to bring together these components into a comprehensive framework. BitDew suits this purpose by providing an environment for data management and distribution in Desktop Grids.

2. THE BITDEW ENVIRONMENT

BitDew is a subsystem which could be easily integrated into other Desktop Grid systems. It offers programmers a simple API for creating, accessing, storing and moving data with ease, even on highly dynamic and volatile environments.

BitDew leverages the use of metadata, a technic widely used in Data Grid, but in more directive style. We define 5 different types of metadata : *i*) *replication* indicates how many occurrences of data should be available at the same time in the system, *ii*) *fault tolerance* controls the resilience of data in presence of machine crash, *iii*) *lifetime* is a duration, absolute or relative to the existence of other data, which indicates when a datum is obsolete, *iv*) *affinity* drives movement of data according to dependency rules, *v*) *transfer protocol* gives the runtime environment hints about the file transfer protocol appropriate to distribute the data. Programmers tag each data with these simple attributes, and simply let the BitDew runtime environment manage operations of data creation, deletion, movement, replication, as well as fault tolerance.

[1]BitDew can be found at `http://www.bitdew.net` under GPL license

Figure 1 illustrates the layered BitDew software architecture upon which distributed applications can be developed.

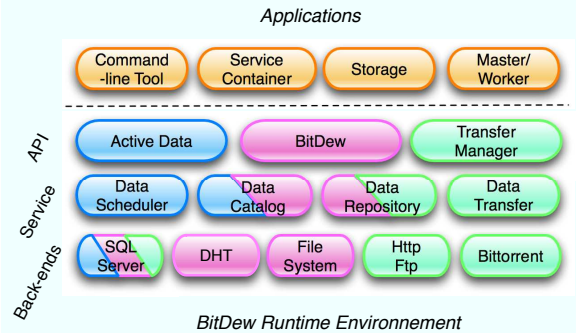

Figure 1: The BitDew software architecture. There are three layers composing the BitDew run-time environment : the API layer, the service layer and the back-ends layer. Colors illustrate how various components of different layers combine together.

The programming model is similar to the Tuple Space model pioneered by Gelernter in the Linda programming system; it aggregates the storage resources and virtualizes it as a unique space where data are stored. The BitDew APIs provide functions to create a slot in this space and to put and get files between the local storage and the data space. The environment provides programmers event-driven programming facilities to react to the main data life-cycle events: creation, copy and deletion. Finally the TransferManager API offers a non-blocking interface to concurrent file transfers, allowing users to probe for transfer, to wait for transfer completion, to create barriers and to tune the level of transfers concurrency.

The intermediate level is the service layer which implements the API : data storage and transfers, replicas and volatility management. The architecture follows a classical approach commonly found in Desktop Grids: it divides the world in two sets of nodes : stable nodes and volatile nodes. Stable nodes, or *service hosts*, run various independent services which compose the runtime environment: Data Repository (DR), Data Catalog (DC), Data Transfer (DT) and Data Scheduler (DS). Volatile nodes can either ask for storage resources (we call them *client hosts*) or offer their local storage (they are called *reservoir hosts*). Classically in DG, we use a *pull model*, where volatile nodes periodically contact service nodes to obtain data and synchronize their local data cache.

The lowermost level is composed of a suite of back ends. The Bitdew runtime environment delegates a large number of operations to third party components : 1) Meta-data information are serialized using a traditional SQL database, 2) data transfers are realized out-of-band by specialized file transfer protocols, such as BitTorrent or FTP/Http, and 3) publish and look-up of data replicas are enabled by the means of DHT protocols. One feature of the system is that all of these components can be replaced and plugged-in by the users, allowing them to select the most suitable subsystem according to their own criteria like performance, reliability and scalability.

3. PROGRAMING A MASTER/WORKER APPLICATION

As an example, we present a data-intensive Master/worker application developped with BitDew. The application is based on NCBI BLAST (Basic Local Alignment Search Tool), BLAST compares a query sequence with a database of sequences, and identifies library sequences that resemble the query sequence above a certain threshold.

In a classical MW application, tasks are created by the master and scheduled to the workers. Once a task is scheduled, the worker has to download the data needed before the task is executed. In contrast, the data-driven approach followed by BitDew implies that data are first scheduled to hosts. The programmer do not have to code explicitly the data movement from host to host, neither to manage fault tolerance. Programming the master or the worker consists in operating on data and attributes and reacting on data copy.

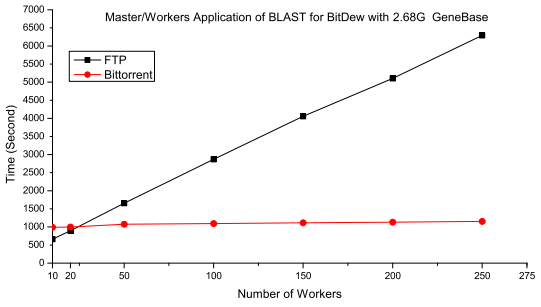

Figure 2: BitDew performances on a Master/Worker application. The two lines present the average total execution time in seconds for the BLAST application with a large Genebase of 2.68GB, executed on 10 to 250 nodes and with file transfer performed by FTP and BitTorrent

4. REFERENCES

[1] David Anderson. BOINC: A System for Public-Resource Computing and Storage. In *proceedings of the 5th IEEE/ACM International GRID Workshop*, Pittsburgh, USA, 2004.

[2] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, 2003.

[3] Gilles Fedak, Cecile Germain, Vincent Neri, and Franck Cappello. XtremWeb: A Generic Global Computing Platform. In *CCGRID'2001 Special Session Global Computing on Personal Devices*, 2001.

[4] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. In *Proceedings of IEEE/INFOCOM 2005*, Miami, USA, March 2005.

[5] John Kubiatowicz and all. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[6] S. Vazhkudai, X. Ma, V. Freeh, J. Strickland, N. Tammineedi, and S. L. Scott. FreeLoader: Scavenging Desktop Storage Resources for Scientific Data. In *SC'05*, Seattle, 2005.