BitDew: A Programmable Environment for Large-Scale Data Management and Distribution

Gilles Fedak, Haiwu He, Franck Cappello

Grand-Large/INRIA-Futurs
Laboratoire de Recherche en Informatique
Université Paris XI

Innovative Computing Laboratory,
Knoxville 2008

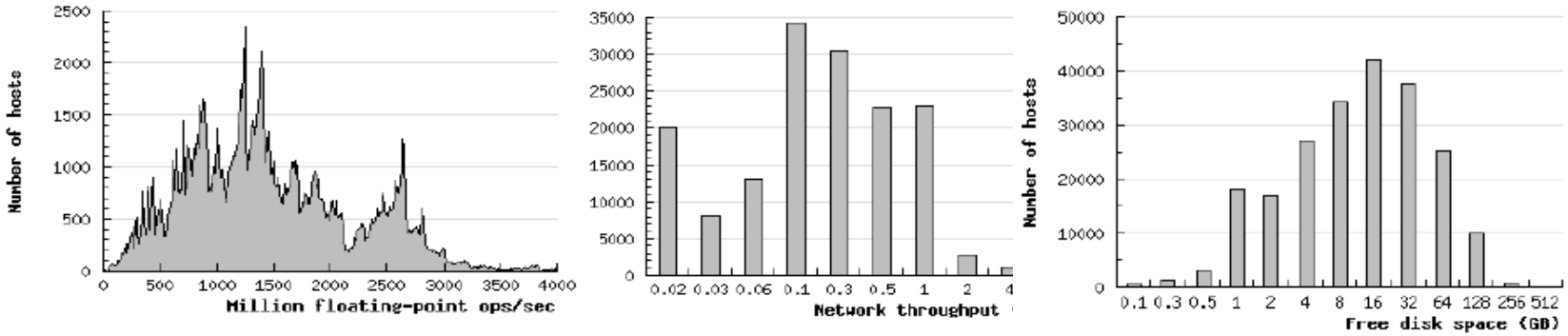
Outline of Topics

Outline

Introduction to Computational Desktop Grids

High Throughput Computing over Large Sets of Idle Desktop Computers

- Internet Distributed Computing (SETI@Home, Distributed.net)
- Open Source Desktop Grid (BOINC, XtremWeb, Ourgrid, Condor etc. . .)



- Scalable but mainly for embarrassingly parallel applications with few I/O requirements
- Challenge is to broaden the application domain

Challenge of Data Management for Desktop Grids

Characteristics of Desktop Grids Resources

- High number of resources
- Volatility
- Low performance
- Owned by volunteer

We've looked at several classes of challenging applications, not yet supported on Desktop Grids, and examine their requirements in term of data management.

- Data-intense parameter sweeps application
- Long running applications
- Workflows applications
- Soft real-time, data stream processing

Can P2P Technologies Help ?

- Collaborative Content Distribution Network (BitTorrent, Avalanche)
- Distributed Hash Table (Chord, Kademlia, Pastry)
- Wide-area and deep Storage (IBP, Oceanstore)
- Storage over volatile resources (Farsite, Freeloader)

- one needs to bring together these components into a comprehensive framework

What is BitDew ?

BitDew : a Programmable Environment for Large Scale Data Management

Key Idea 1: provides an API and a runtime environment which integrates several P2P technologies in a consistent way

Key Idea 2: relies on metadata (*Data Attributes*) to drive transparently data management operation : replication, fault-tolerance, distribution, placement, life-cycle.

Outline

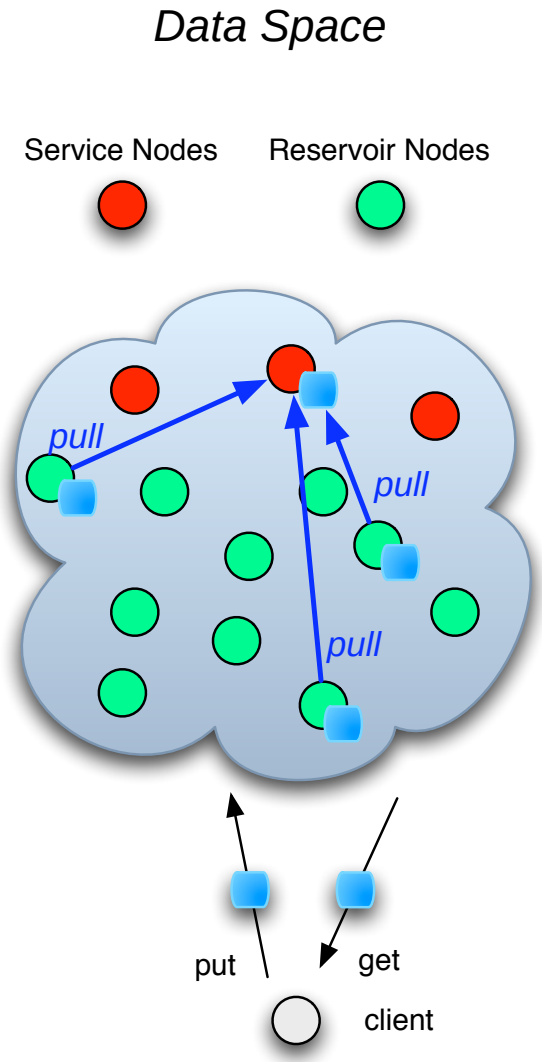
BitDew : the Big Cloudy Picture

- Aggregates storage in a single Data Space:
  - Clients `put` and `get` data from the data space
  - Clients defines data attributes
- Distinguishes *service* nodes (stable), *client* and *reservoir* nodes (volatile)
  - *Service* : ensure fault tolerance, indexing and scheduling of data to *reservoir* nodes
  - *Reservoir* : stores data on Desktop PCs
- *push/pull* protocol between client → service ← reservoir

BitDew : the Big Cloudy Picture

- Aggregates storage in a single Data Space:
  - Clients `put` and `get` data from the data space
  - Clients defines data attributes

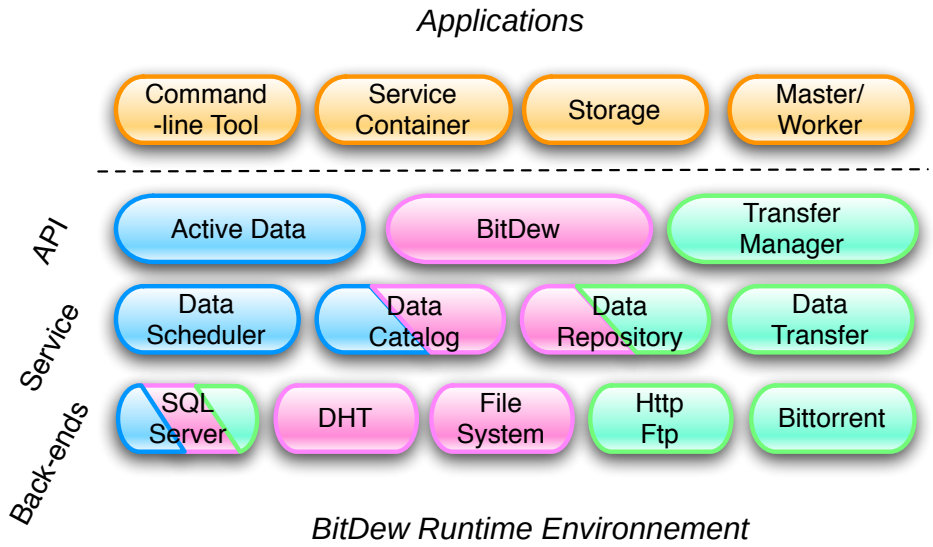
- Distinguishes *service* nodes (stable), *client* and *reservoir* nodes (volatile)
  - *Service* : ensure fault tolerance, indexing and scheduling of data to *reservoir* nodes
  - *Reservoir* : stores data on Desktop PCs
- *push/pull* protocol between client $\longrightarrow$ service $\longleftarrow$ reservoir



Data Attributes

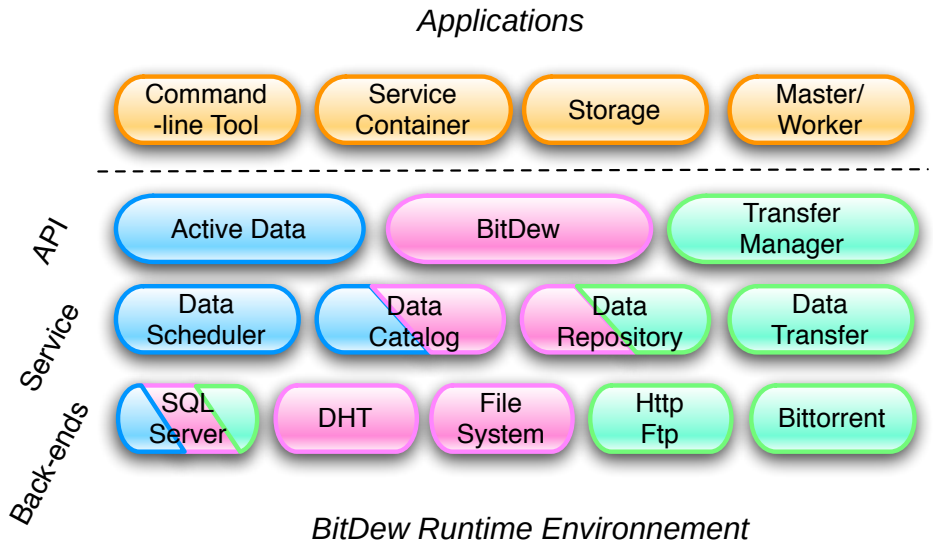
REPLICA :	indicates how many occurrences of data should be available at the same time in the system
FAULT-TOLERANCE :	controls the resilience of data in presence of machine crash
LIFETIME :	is a duration, absolute or relative to the existence of other data, which indicates when a datum is obsolete
AFFINITY :	drives movement of data according to dependency rules
TRANSFER PROTOCOL :	gives the runtime environment hints about the file transfer protocol appropriate to distribute the data

Architecture Overview



- 3 layers : API, Services and Back-ends
- Programmers only use the API level
- Services never communicate together

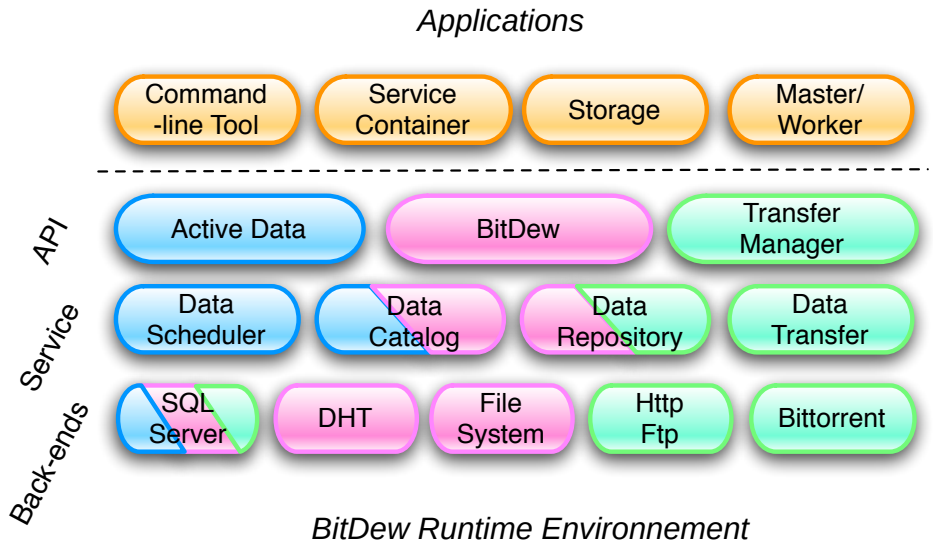
Architecture Overview



The Uppermost level : the API

- *BitDew* : provides functions to create slots in the Data Space and to put and get files between the local storage and the slots
- *ActiveData* : manages Data Attribute, schedules Data and provides programmers an event-driven programming facilities to react to the main data life-cycle events: creation, copy and deletion
- *TransferManager* : a non-blocking interface to concurrent file transfers

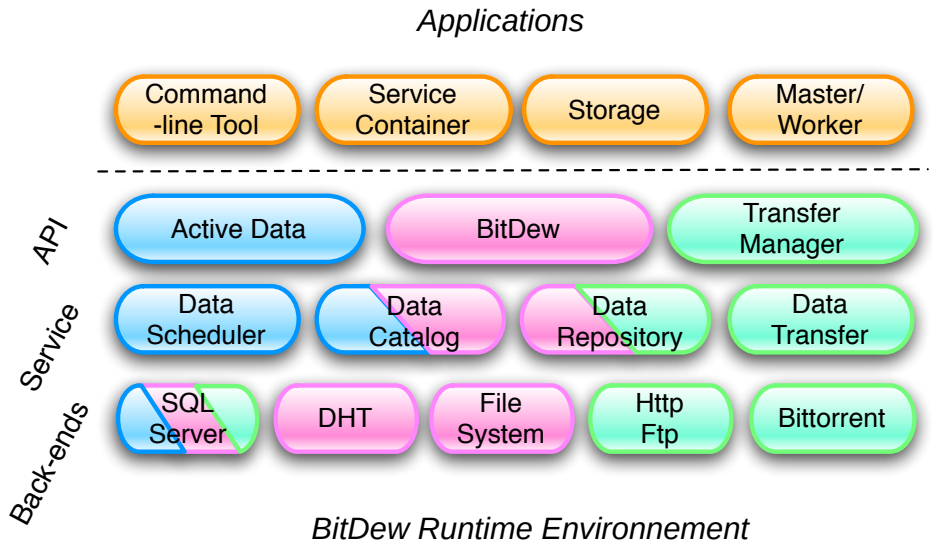
Architecture Overview



The Intermediate level : the Services

- Data Catalog (DC) : stable index of permanent copy of data
- Data Repository (DR) : stores data and provides remote access to the data
- Data Transfer (DT) : ensures reliability of out-of-band file transfer protocol
- Data Scheduler (DS) : schedules data on reservoir nodes and ensures fault-tolerance of data

Architecture Overview



The Lowermost Level : the Backend System

- Time consuming operations are delegated : 1) Meta-data information are serialized with SQL DB, 2) data transfer by out-of-band file transfer protocol FTP/BitTorrent, 3) Distributed Data Catalog (DDC) with a DHT
- Permanent copies of data (DR) are indexed in the DC but data replica on volatile are indexed in the DDC

Implementation

Implementation of the prototype

- Core communication: Java RMI
- Core serialization : Java JDO (`http://jakarta.apache.org`) stack with JPOX (`http://jpox.org`)
- DB: MySQL (`http://mysql.com`)/HsqlDB (`http://hsqldb.org`)
- File Transfer : WU-FTPD (`http://wu-ftpd.org`), Azureus BitTorrent client (`http://azureus.sf.net`)
- DHT : DKS DHT (`http://dks.sics.se/`)

Less than 18000 lines of code.
First releases are available at `http://www.bitdew.net` under GNU GPL

Outline

Experiment Setup

Hardware Setup

- Experiments are conducted in a controlled environment (cluster for the microbenchmark and Grid5K for scalability tests) as well as an Internet platform
- *GdX Cluster* : 310-nodes cluster of dual opteron CPU (AMD-64 2Ghz, 2GB SDRAM), network 1Gbs Ethernet switches
- *Grid5K* : an experimental Grid of 4 clusters including GdX
- *DSL-lab* : Low-power, low-noise cluster hosted by real Internet users on DSL broadband network. 20 Mini-ITX nodes, Pentium-M 1Ghz, 512MB SDRam, with 2GB Flash storage
- Each experiments is averaged over 30 measures

Core Data Operation

	without DBCP		with DBCP	
	MySQL	HsqlDB	MySQL	HsqlDB
local	O.25	3.2	1.9	4.3
RMI local	0.21	2.0	1.5	2.8
RMI remote	0.22	1.7	1.3	2.1

Table: Numbers are expressed as thousands of data creation per second

Performance of Basic Data Operations

- Benchmark is a client running a loop which continuously creates data slot in the storage space, and a server running the Data Catalog service.
- Representative of critical path of core operation: One RMI call + 1 SQL INSERT or UPDATE
- Latency is 500μsec, but it can be enhanced using bursted and multithreaded requests.

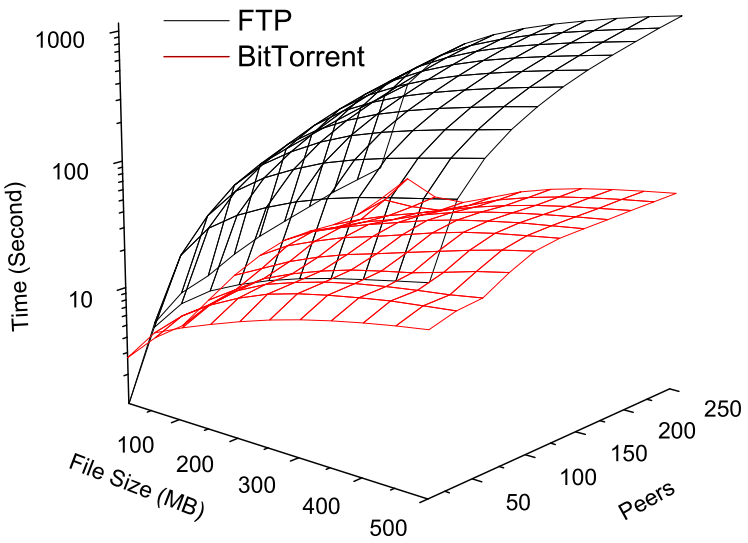
DC vs DDC

	Min	Max	Sd	Mean
publish/DDC	100.71	121.56	3.18	108.75
publish/DC	2.20	22.9	5.05	7.02

Table: Performance evaluation of data publishing in the centralized and distributed data catalog : the numbers represents the pairs (dataID,hostID) created per second.

- The benchmark consists of an SPMD program running on 50 nodes. After a synchronisation, each node will publish 500 pairs of dataID, hostID values
- DDC is around 15 time slower than DC to index the 25000 data

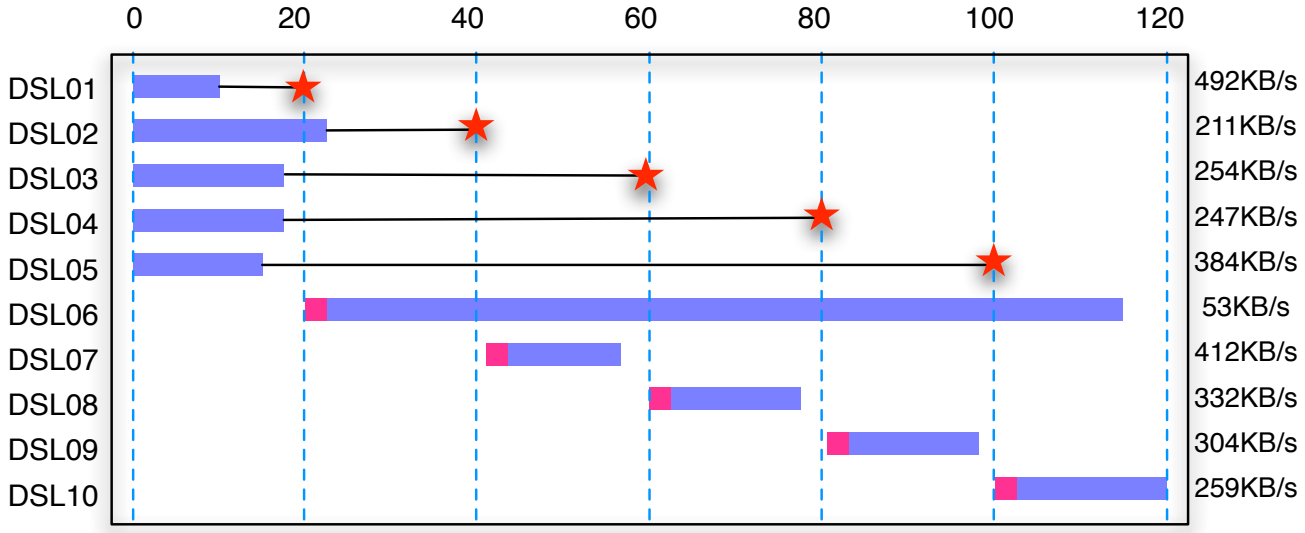
Out-of-Band File Transfers



File Distribution with BitTorrent vs FTP

- Small files (10K to 50MB):
  BitTorrent presents a higher overhead than FTP.
- Large Number of nodes ($>$ 40):
  FTP latency grows linearly as the bandwidth is shared among downloaders.
  BitTorrent is scalable due to the efficiency of nodes cooperation.

Fault-Tolerance Scenario

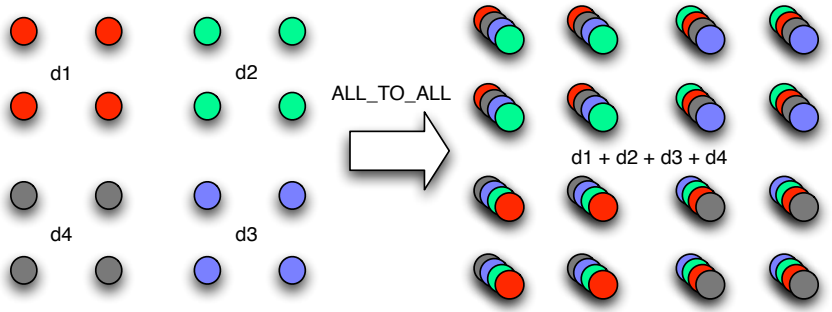


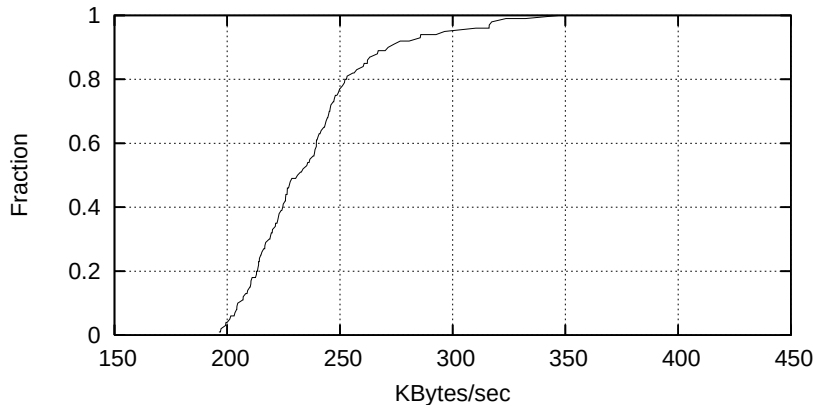
Evaluation of Bitdew in Presence of Host Failures

Scenario: a datum is created with the attribute : REPLICA = 5, FAULT_TOLERANCE = true and PROTOCOL = "ftp". Every 20 seconds, we simulate a machine crash by killing the BitDew process on one machine owning the data. We run the experiment in the DSL-Lab environment

Figure: The Gantt chart presents the main events : the blue box shows the download duration, red box is the waiting time, and red star indicates a node crash.

Collective File Transfer with Replication





Scenario: All-to-all file transfer with replication.

- $\{d1, d2, d3, d4\}$ is created where each datum has the REPLICA attribute set to 4
- All-to-all collective implemented with the AFFINITY attribute. Each data attributes is modified on the fly so that *d1.affinity=d2, d2.affinity=d3, d3.affinity=d4, d4.affinity=d1*.

Results: experiments on DSL-Lab

- one run every hour during approximately 12 hours
- during the collective, file transfers are performed concurrently and the data size is 5MB.
- Cumulative Distribution Function (CDF) plot for collective all-to-all.

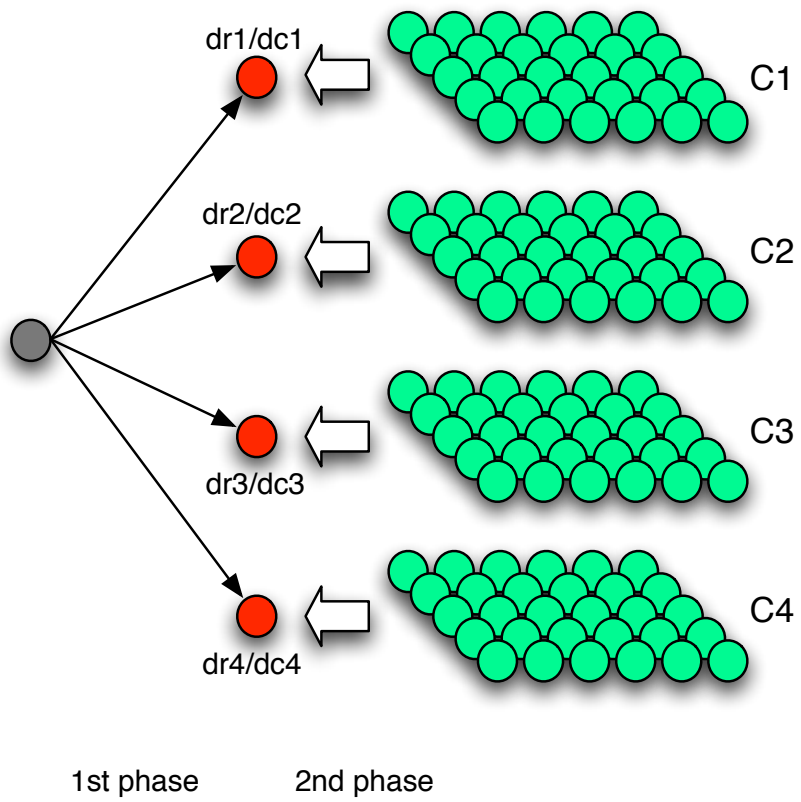
Multi-sources and Multi-protocols File Distribution

Scenario

- 4 clusters of 50 nodes each, each cluster hosting its own data repository (DC/DR).
- first phase: a client uploads a large datum to the 4 data repositories
- second phase: the cluster nodes get the datum from the data repository of their cluster.

We vary:

1. the number of data repositories from 1 to 4
2. the protocol used to distribute the data (BitTorrent vs FTP)



Multi-sources and Multi-protocols File Distribution

	BitTorrent (MB/s)					FTP (MB/s)			
	orsay	nancy	bordeaux	rennes	mean	orsay	nancy	bordeaux	rennes
1st phase									
1DR/1DC	30.34				30.34	98.50			
2DR/2DC	29.79		12.35		21.07	73.85		49.86	
4DR/4DC	22.07	10.36	11.51	9.72	13.41	49.41	22.95	26.19	22.89
2nd phase									
1DR/1DC	3.93	3.93	3.78	3.96	3.90	0.72	0.70	0.56	0.60
2DR/2DC	4.04	4.88	9.50	5.04	5.86	1.43	1.38	1.14	1.29
4DR/4DC	8.46	7.58	7.65	5.14	7.21	2.90	2.78	2.33	2.54

- for both protocols, increasing the number of data sources also increases the available bandwidth
- FTP performs better in the first phase and BitTorrent gives superior performances in the second phase
- the best combination (4DC/4DR, FTP for the first phase and BitTorrent for the second phase) takes 26.46 seconds and outperforms by 168.6% the best single data source, single protocol (BitTorrent) configuration

Programming a Master/Worker Application

Implementing BLAST on BitDew

- BLAST : (Basic Local Alignment Search Tool) compares a query sequence with a database of sequences
- Data-intense Bag-of-Tasks
- Data-driven Master/Worker : data are scheduled first. When the *Application*, the *Database* and a *Sequence* are present on a node, a computation is launched and a *Result* is produced

```
attribute Application = { replica = -1, protocol = "BitTorrent"
}
attribute Sequence = {  protocol = "http"
}
attribute Genebase = { protocol = "BitTorrent", affinity = Sequence
}
attribute Result = { protocol = "http"
}
```

Programming a Master/Worker Application

Implementing BLAST on BitDew

- BLAST : (Basic Local Alignment Search Tool) compares a query sequence with a database of sequences
- Data-intense Bag-of-Tasks
- Data-driven Master/Worker : data are scheduled first. When the *Application*, the *Database* and a *Sequence* are present on a node, a computation is launched and a *Result* is produced

```
attribute Application = { replica = -1, protocol = "BitTorrent"
}
attribute Sequence = {  protocol = "http", lifetime = Collector,
}
attribute Genebase = { protocol = "BitTorrent", affinity = Sequence, lifetime =
     Collector,
}
attribute Result = { protocol = "http", affinity = Collector, lifetime =
     Collector
}
attribute Collector = {
}
```

Programming a Master/Worker Application

Implementing BLAST on BitDew

- BLAST : (Basic Local Alignment Search Tool) compares a query sequence with a database of sequences
- Data-intense Bag-of-Tasks
- Data-driven Master/Worker : data are scheduled first. When the *Application*, the *Database* and a *Sequence* are present on a node, a computation is launched and a *Result* is produced

```
attribute Application = { replica = -1, protocol = "BitTorrent"
}
attribute Sequence = {  protocol = "http", lifetime = Collector, fault tolerance
     = true, replication = x
}
attribute Genebase = { protocol = "BitTorrent", affinity = Sequence, lifetime =
     Collector,
}
attribute Result = { protocol = "http", affinity = Collector, lifetime =
     Collector
}
attribute Collector = {
}
```

Performance Evaluation

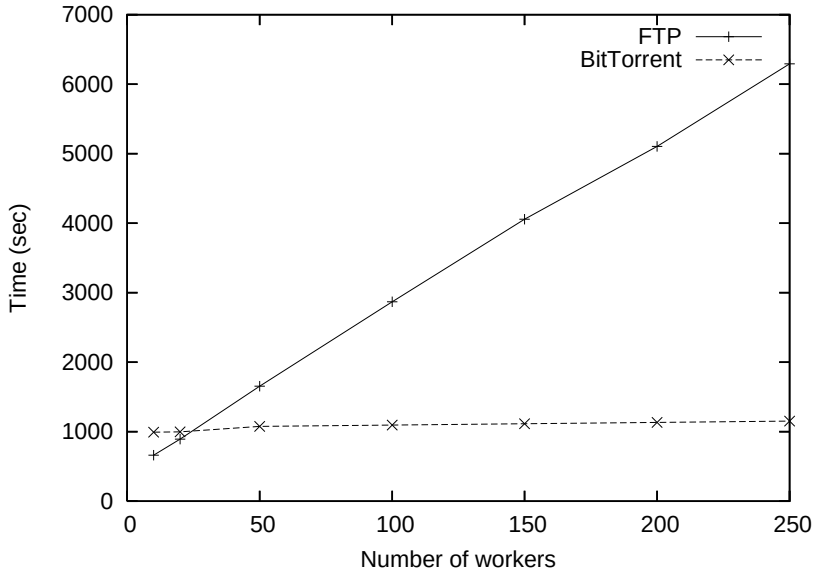


Figure: Scalability evaluation: the two lines present the average total execution time in seconds for the BLAST application, executed on 10 to 250 nodes

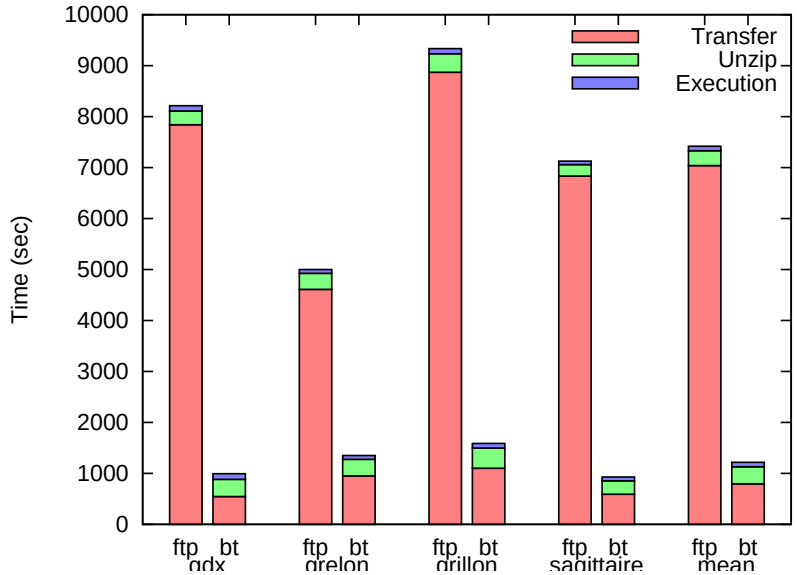


Figure: Breakdown of total execution time in time to transfer data, time to unzip data and Blast execution time by cluster. The experiment uses 400 nodes distributed over 4 clusters. The rightmost values is the time average on the whole platform.

Outline

Conclusion

- BitDew :
  - leverages metadata (*Data Attributes*) to drive transparently data management operation : replication, fault-tolerance, distribution, placement, life-cycle
  - features multi-protocols file transfer, data scheduling, automatic replication and transparent data placement
- Use cases :
  - automatic replication in case of host failures
  - all-to-all file transfer with replicated data
  - multi-sources and multi-protocols file distribution
  - data-driven master/worker application on a data-intense BoT application (BLAST)
- Future works :
  - Data Desktop Grids : sliced data, collective communication such as gather/ scatter, and other programming abstractions, such as support for distributed MapReduce operations.

Thank you !