- Behind The Cloud -

Infrastructure and Technologies used for Cloud Computing

Alexander Huemer, 0025380
Johann Taferl, 0320039
Florian Landolt, 0420673

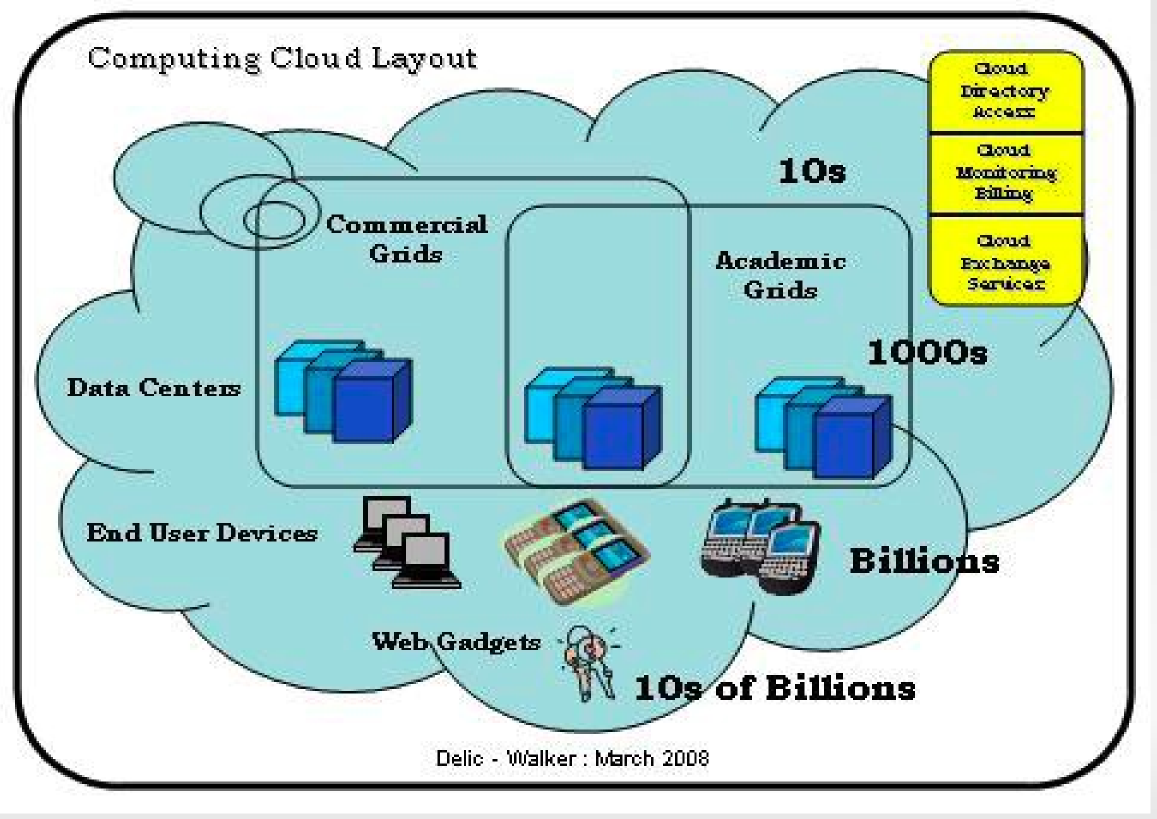
Seminar aus Informatik, University of Salzburg

Overview

1. Definition
   - Cloud Computing vs. Grid Computing
2. Infrastructure
   - Distributed FS
     - Andrew FS
     - Google FS
   - Distributed Scheduling
     - Condor
   - Distributed Applications
     - Google Map/Reduce

Definition

- ”Computing clouds are huge aggregates of various grids (academic, commercial), computing clusters and supercomputers.” [6]



Distributed FS

- Definition
- Examples
  - Andrew FS
  - Google FS
- Comparison

Distributed Filesystems

- Allow multiple hosts to access (r/w) block data as though it would be available locally.
- Enable multiple users (applications) on multiple machines to share block data.
- Are not shared disk filesystems (cluster filesystems)

Main characteristics

- Client nodes do not have direct block access to the storage
- Interaction enabled via a high level network ({l,w,m}an) protocol.
- Access is abstracted, use is not distinguished from local filesystems
- Access restrictions can be implemented in the network protocol, not necessarily equal to local fs rights.

High availability

- Features like transparent replication and fault tolerance may be but do not have to be implemented
- A limited number of back end storage servers may fail, but the filesystem remains available
- A central Head-Server is common – it may not fail under any circumstances

Popular nonCloud Examples

- NFS (Network File System)
- SMB/CIFS (Windows File Sharing & Samba)

Cloud-relevant Filesystems

- Google Filesystem (GFS)
- Andrew Filesystem (AFS)

Andrew Filesystem

- Created at Carnegie Mellon University
- Fully supported by the Linux kernel
- Free user space implementations exist

Andrew Filesystem (cont'd)

- Kerberos secured network transport
- Intelligent client side caching allows limited offline usage
- Quotas can be used to manage space usage

Andrew Filesystem (cont'd)

- Unique namespace and security features allow publishing of AFS services on untrusted networks like the internet
- Actual volume storage can be moved without the need to notify users, even with open files
- Read-only clones can be created
- Large Files (over 4GB) support is matured

Google Filesystem

- Designed for Google internal usage
- Currently not freely available
- Former “BigFiles” when Google was still in Stanford
- Fixed Chunk size of 64 Mbyte
- Running on a large number of cheap nodes
- Master Node is nevertheless necessary

Google Filesystem (cont'd)

- Master node does not store the content of the files, only metadata
- A defined number of copies of each chunk is actively maintained
- Communication between Chunk server by simple heartbeats
- After metadata retrieval, clients transfer actual data directly to/from chunk servers

Google Filesystem (cont'd)

- Implementation completely in user space
- Not all aspects of a classic filesystem are honored

Comparison (AFS vs. GFS)

- AFS is designed to run on dedicated server hardware, parallelism is implemented for redundancy in the classical sense
- GFS has explicit support for unreliable hardware

Comparison (AFS vs. GFS)

- GFS has very sophisticated features but only for a rather limited range of applications
- AFS is not tuned for a particular purpose, but can be used for a wide range of fileserving needs

Distributed Scheduling

- Definition / Types
- Condor
  - Features
  - ClassAds
  - Universes

DS – Definition / Types

Algorithms

- Concentrate on policy

Scheduling Systems

- Provide mechanism to implement algorithms
  - Inside and outside the OS
  - Distributed (message passing) and parallel (shared memory)
  - Space sharing (n processors to 1 job) and time sharing (n tasks to 1 processor)
  - etc.

DS – Condor

- User submit jobs
- Condor
  - places them in a queue
  - chooses when and where to execute them
  - monitors progress
  - informs user upon completion

”Condor can be used to seamlessly combine all of your organization's computational power into one resource”

DS – Condor – Features

- Distributed Submissions
- Job Priorities
- User Priorities
- Job Dependence
- Support for Multiple Job Models
- **ClassAds**
- Job Checkpoint and Migration
- Periodic Checkpoint
- Job Suspend and Resume
- Remote System Calls

DS – Condor – Features (cont'd)

- Pools of Machines can Work Together
- Authentication and Authorization
- Heterogenous Platforms
- Grid Computing

DS – Condor – ClassAds

Can be compared to advertising in the Newspaper

- Sellers advertise specifics about what they want to sell
- Buyers advertise specifics about what they purchase

Sellers = Machine owners / resources
Buyers = Users submitting jobs

DS – Condor – ClassAds (cont'd)

- Structure of a ClassAd
  - Set of unique named expressions (attribute name = attribute value)

```
Memory = 512
OpSys = "LINUX"
NetworkLatenc = 7.5
```

DS – Condor – ClassAds (cont'd)

- Attribute values can consist of logical expressions which are evaluated in terms of:
  - TRUE
  - FALSE
  - UNDEFINED

DS – Condor – ClassAds (cont'd)

- Attribute values can consist of logical expressions which are evaluated in terms of:
  - TRUE
  - FALSE
  - UNDEFINED

```
MemoryInMegs = 512
MemoryInBytes = MemoryInMegs * 1024 * 1024
Cpus = 4
BigMachine = (MemoryInMegs > 256) && (Cpus >= 4)
VeryBigMachine = (MemoryInMegs > 512) && (Cpus >= 8)
FastMachine = BigMachine && SpeedRating
```

DS – Condor – ClassAds (cont'd)

- Matching ClassAds
  - Job ClassAd

```
MyType = ″Job″
TargetType =″Machine″
Requirements = ((Arch==″INTEL″ && OpSys==″″LINUX)
&& Disk > DiskUsage)
Rank = (Memory * 10000) + Kflops
…
```

  - Machine ClassAd

```
MyType = ″Machine″
TargetType = ″Job″
Requirements = Start
Rank = TARGET.Department == MY.Department
Activity = ″Idle″
Arch = ″INTEL″
...
```

DS – Condor – Universes

- Universe = Runtime Environments
- Determined by type of application
- Six job Universes in total
  - Standard
  - Vanilla
  - PVM
  - MPI
  - Globus
  - Scheduler

DS – Condor – Universes (cont'd)

- Vanilla Universe
  - Used to run serial (non-parallel) jobs
  - Not restricted at all
  - Relies on shared filesystem

DS – Condor – Universes (cont'd)

- MPI Universe
  - Parallel programs written with MPI (Message Passing Interface)
  - Number of nodes for parallel job are needed (attribute `machine_count`)
  - Other attributes are supported
    - Job requirements
    - Executable across different nodes
    - etc.

DS – Condor – Universes (cont'd)

- PVM Universe
  - parallel programs in Master-Worker style
    (written for Parallel-Virtual-Machine interface)
    - Master gets pool of tasks and sends peaces work out to the worker nodes
  - Can harness dedicated and non-dedicated workstations by adding and removing machines to and from the PVM
  - Condor starts master application
    Worker jobs are pulled in from pool as they become available
  - MW tool helps to generate Master-Worker applications

DS – Condor – Universes (cont'd)

- Globus Universe
  - Machines are managed by Globus (http://www.globus.at/)

DS – Condor – Universes (cont'd)

- Scheduler Universe
  - Job will immediately run on the submit machine (as opposed to a remote executing machine)
  - Meta-Schedulers provide submission and removal of jobs into Condor queue.
  - Implementation: DAGMan
    - Direct Acyclic Graph
    - For complicated dependencies

DS – Condor – Universes (cont'd)

- Standard Universe
  - Minimal extra effort on the user's part
  - Provides a serial job with services
    - Transparent process checkpoint and restart
    - Transparent process migration
    - Remote System Calls
    - Configurable I/O buffering
    - On-the-fly file compression/inflation

MapReduce

- Definition
- Concept
- Example
- Implementation

MapReduce - Definition

is an abstraction which allows writing distributed code without having to deal with

- parallelization
- scalability
- fault-tolerance
- data distribution
- load balancing

MapReduce - Concept

- **Map function:**
  processes a key/value pair and generates a set of intermediate key/value pairs
- **Reduce function:**
  merges all intermediate values associated with the same intermediate key to a set of values which is possibly smaller

map(k,v)

↓

list(i_k, i_v)

reduce(i_k, list(i_v))

↓

list(i_v)

MapReduce - Example

Counting the number of occurrences of each word in a large collection:

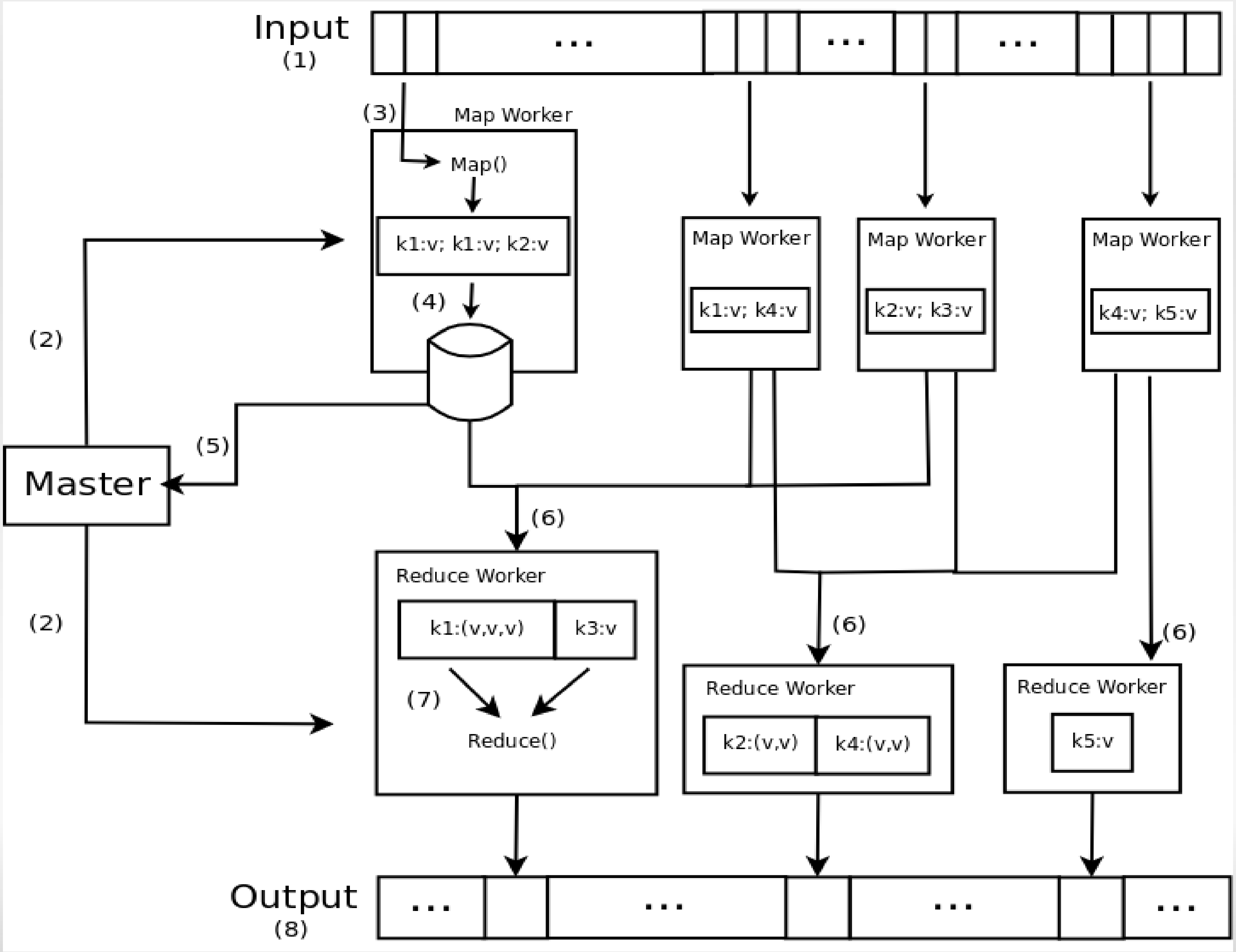
```
map(String key, String value):
   //key: document name
   //value: document contents
   for each word w in value:
      EmitIntermediate(w, ”1”);

reduce(String key, Iterator values):
   //key: a word
   //values: a list of counts
   int result = 0;
   for each v in values:
      result += ParseInt(v);
   Emit(AsString(result));
```

MapReduce - Implementation

3 different kinds of nodes:

- Master (only one)
- Map worker
- Reduce worker

MapReduce – Implementation (cont'd)



Bibliography

(1) J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, January 2008.

(2) S. J. Chapin. Distributed and multiprocessor scheduling. *ACM Comput. Surv.*, 28(1):233–235, 1996.

(3) S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

(4) M. Satyanarayanan and M. Spaso jevic. Afs and the web: competitors or collaborators? *In EW 7: Proceedings of the 7th workshop on ACM SIGOPS European workshop*, pages 89–94, New York, NY, USA, 1996. ACM.

(5) T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor – a distributed job scheduler.

(6) K. A. Delic and M. A. Walker. Emergence of the academic computing clouds. Ubiquity, 9(31):1–1, 2008.