CloudI

A Cloud as an Interface

EXPANDED

Erlang Factory SF Bay Area, California, USA
March 26, 2010

Michael Truog
mjtruog@gmail.com

Cloudi Terminology

- Work Title == “Work Module.Tag”
- Data Title == “Data Module.Database”
- Worker – a work thread within the `cloud_worker_port` operating system process connected to Cloudi
- cnode – Erlang node implemented in C
- Erlang port – operating system process spawned by the Erlang VM

Cloudi Topics

1. What is Cloudi?
2. Why Use Cloudi?
3. Where Is Cloudi Used?
4. How To Use Cloudi
5. The Future

What is Cloudi?

1. Private Cloud Computing Framework
2. Fault-tolerant Work Processing
3. Dynamic Load Balancing and Scheduling
4. Ordered Work Input/Output
5. Distributed Execution of C/C++ Work

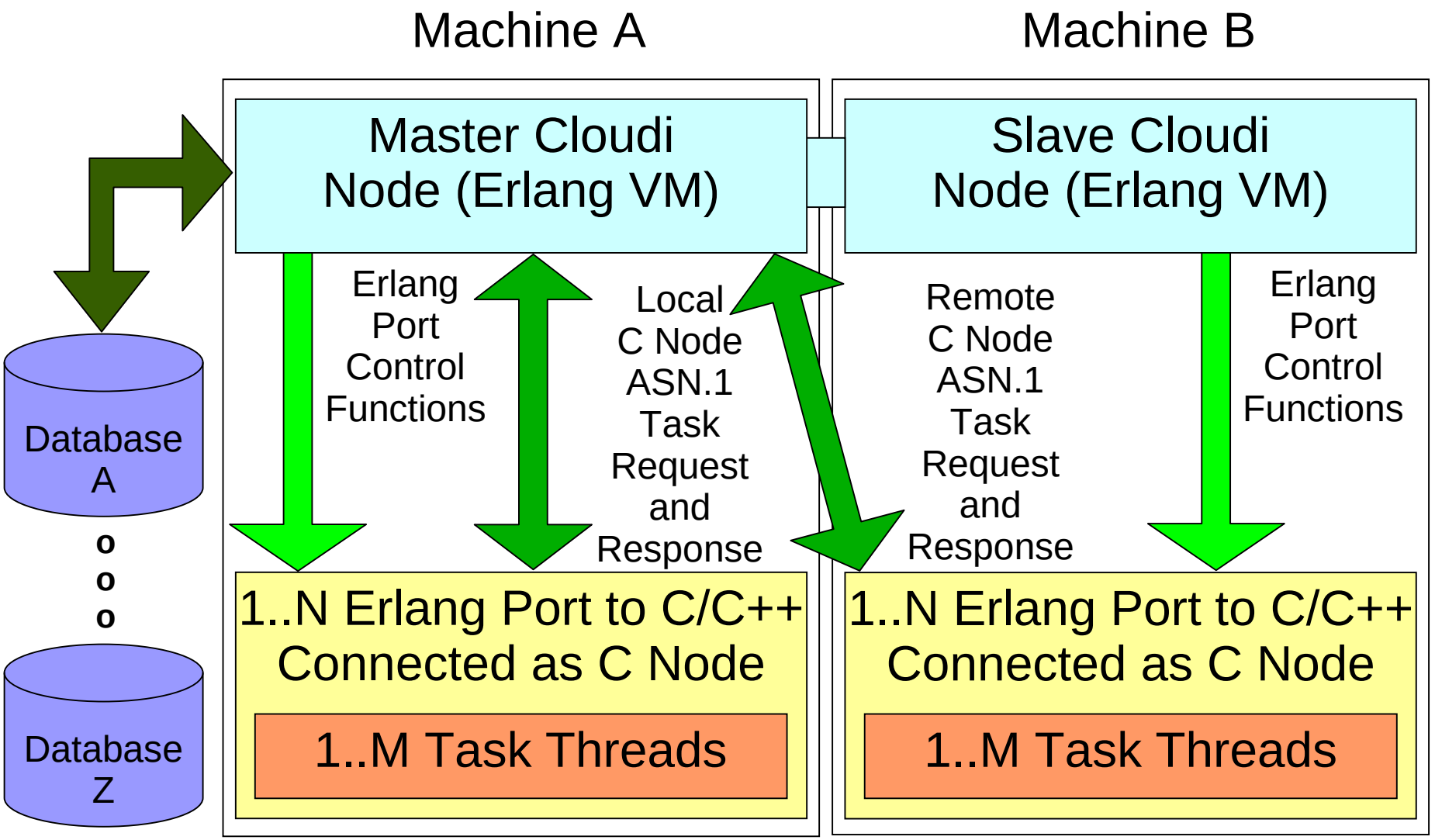
A Private Cloud Computing Framework

- Provides an open-source cloud
  - BSD License
- An alternative to paying for a black-box commercial cloud
  - Internal processing is secure processing
- Creates a stable distributed processing environment from any available Linux machines

Fault-tolerant Work Processing

- Erlang/OTP coordinates all work allocation, execution, and work data flow
- Any crash of C/C++ code is handled
  - Any signals, including uncatchable signals
- Uses Erlang Port processes subscribing to the cloud as Erlang C Nodes
  - Fault-tolerance overhead (“trip1”) averages 0.129 ms/task locally and 0.334 ms/task remotely (http://cloudi.org/latency/latency.html)

Fault-tolerant Work Processing (cont.)



Dynamic Load Balancing and Scheduling

- Worker threads are ideally stateless and form a pool of workers in the cloud
- Cloudi adjusts the task size based on the task execution time that is requested
  - Convergence is slow to avoid problems with unstable work processing
- Cloudi verifies that work is loaded:
  - During work allocation
  - After node reconnection

Ordered Work Input/Output

- The Erlang work module enforces an order on the work task input
- Cloudi maintains the task input order when collecting output so data is stored in the same order
- Work processing is paused when excessive data accumulation occurs

Distributed Execution of C/C++ Work

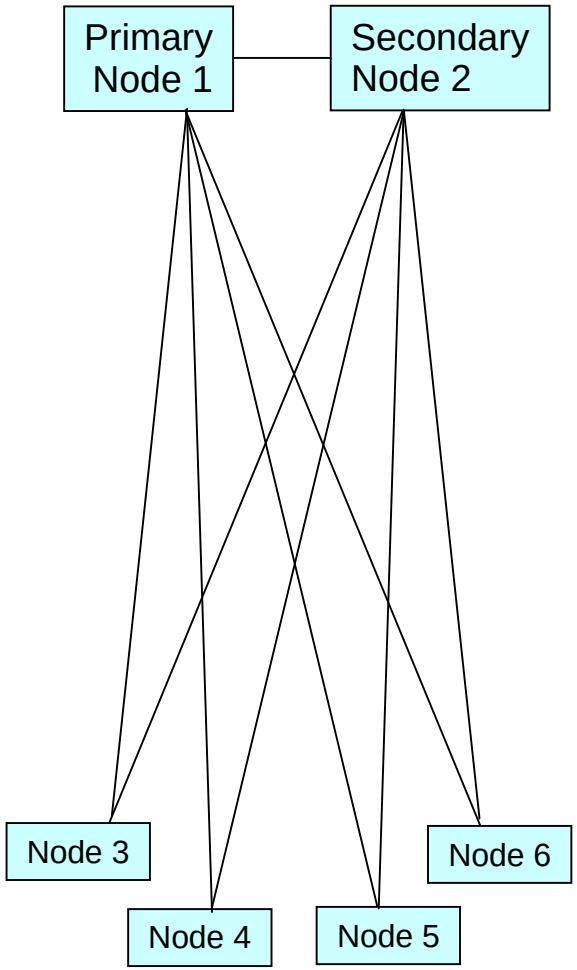
- One `do_work` function is required in a dynamic library for the C/C++ work
  - Loaded when Cloudi requests it
- Six Erlang functions within the work module provide work task specification
  - The functions define the task size as a float value in the range (0..1) and task data as binary data
- Any Erlang data module can handle output
  - Currently the supported databases are PostgreSQL, MySQL, memcached, Tokyo Tyrant, and CouchDB

Why Use Cloudi?

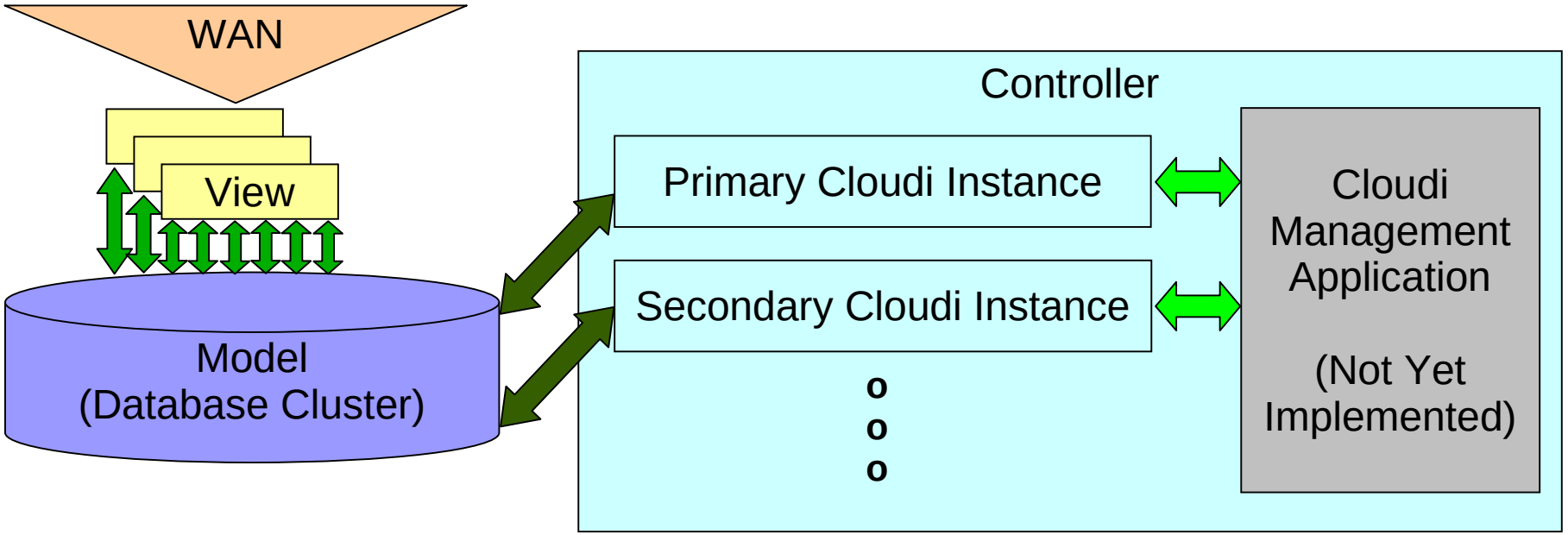
- Computationally intensive data processing
  - Text processing, numerical computations, data transformations, and iterative methods
- Computation is decoupled from external access to the results
  - Separating the computational processes from the resulting data helps to isolate complexity and supports fault-tolerant services

Where Is Cloudi Used?

- A management application can facilitate failover between master nodes
  - Separate epmd processes keep the distributed Erlang nodes separate
- Instance failover can currently be accomplished through manual usage of the `cloud_api` module
  - Not recommended for critical tasks



Where Is Cloudi Used? (cont.)



- Scalability can be achieved with a combination of NoSQL and SQL databases that are clustered

How To Use Cloudi

1. Cloudi Compilation
2. The Erlang Work Module
3. The C/C++ Work Library
4. Cloudi Configuration
5. Cloudi API

Cloudi Compilation

```
make[5]: Leaving directory `/home/user/cloudi-0.0.9/install/g++/releases/gcc-4.4.2/x86_64-unknown-linux-gnu/libgomp'
make[4]: Leaving directory `/home/user/cloudi-0.0.9/install/g++/releases/gcc-4.4.2/x86_64-unknown-linux-gnu/libgomp'
make[3]: Leaving directory `/home/user/cloudi-0.0.9/install/g++/releases/gcc-4.4.2/x86_64-unknown-linux-gnu/libgomp'
make[2]: Leaving directory `/home/user/cloudi-0.0.9/install/g++/releases/gcc-4.4.2'
make[1]: Leaving directory `/home/user/cloudi-0.0.9/install/g++/releases/gcc-4.4.2'
################################################
# g++ NOW BUILT, RERUN THE MAKEFILE TO COMPILE #
################################################
make: *** [/home/user/cloudi-0.0.9/src/lib/g++/releases/gcc-4.4.2_install/bin/g++] Error 1
user@machine$
```

- Compiles g++/gcc locally for all dependencies but takes a lot of time and memory
  - More than 2 hours of compilation time
  - Approximately 3 gigabytes of hard disk storage
  - Only done the first time Cloudi is compiled
- Keeps the Cloudi alpha release maintainable and consistent for diagnosing or reporting problems

The Erlang Work Module

- Uses the `cloud_work_interface` behavior
  - `handle_get_task_time_target/0` controls the smallest interval of job output to the database
  - `handle_get_initial_task_size/0` provides the smallest possible task size for the algorithm
  - `handle_get_task/3` takes the task size and returns the binary task data with the task input database queries that must be processed by the work library
  - `start_link/2` takes job configuration arguments that define the scope of several tasks

The Erlang Work Module (cont.)

- The work module must dynamically adjust the task data in a meaningful way to avoid overloading the database(s)
- Task data must be less than 4 megabytes
- The work module must use the same name as the corresponding C/C++ work library
- A “work title” identifier is the work module name with a unique “.tag” suffix that identifies the type of tasks being processed

The C/C++ Work Library

- Uses the `cloud_work_interface` header file to define the `do_work` function
  - Provides the worker thread id for caching with global work library data
  - The “stop” boolean input parameter changes to make the running task abort its computation
  - A vector of output database queries stores the result of a `do_work` function evaluation which was directly influenced by the task data input parameter created in the Erlang work module

The C/C++ Work Library (cont.)

- Any data repositories must be configured with a “data title” so that output queries are not discarded as irrelevant
- Cloudi depends on a locally compiled version of g++/gcc so that work executes in a consistent environment
- The execution time of the `do_work` function will adjust for tasks in an attempt to converge on the task time target

Cloudi Configuration: Machines

- Machines specification
  - Defines the Cloudi nodes for an instance
  - Specifies the number of operating system processes to use for executing any work and how many threads to allow per process
  - Uses boost::thread to provide threading which encapsulates the pthread API on Linux
  - Specifies port numbers used for each operating system process

Cloudi Configuration: Data

- Data repository specification
  - Database specific settings where a “data title” is a data module name with a “.database” suffix to uniquely identify data routing
  - Startup requires that all databases specified are online
  - The master node for the active instance will die if the database connection is terminated or experiences a timeout

Cloudi Configuration: Jobs

- Jobs specification
  - Every entry must have a unique “work title”, i.e., a work module with “.tag” suffix
  - Includes a request for a number of workers or uses the ‘all’ atom to use all available
  - Either specifies the atom ‘threads’, ‘no_threads’, or an integer that represents threads per operating system process
  - Provides job parameters as arguments to the work module `start_link/2` function

Cloudi API

- Provides a dynamic configuration for machines, data repositories, and jobs
- Uses the same specification format as used in the cloud.conf configuration file
- Does not block the removal of a data repository that running jobs depend on
- Will be the interface for an external management application
- Exists as the `cloud_api` Erlang module

The Future

- The management application needs to be created to simplify Cloudi instance failover
- More databases will be supported
- More fault-tolerance testing
- Download Cloudi @ http://cloudi.org/
  - Version 0.0.9 alpha is now available!

Questions?