AzureBLAST: A Case Study of Developing Science Applications on the Cloud

Wei Lu, Jared Jackson, Roger Barga
Microsoft Research
BLAST

BLAST (Basic Local Alignment Search Tool)

• The most important software in bioinformatics
  – (“blast” is the daily tool)
• Identify the similarity between bio-sequences (e.g. protein)
• NCBI-BLAST: One of most widely used implementations

BLAST is highly computation-intensive

• Large number of pairwise alignment operations
  – The size of sequence databases has been growing exponentially
    » GenBank doubled in size in about 15 months.
• A normal BLAST running could take 700 ~ 1000 CPU hours
Motivation

It is easy to parallelize BLAST on the cluster
  • Segmenting the input
    – embarrassingly parallel
  • Segmenting the database (e.g., mpiBLAST)

Large volume data
  • A normal Blast database can be as large as 10GB
  • The output of BLAST is usually 10-100x larger than the input

For most biologists, two choices for running large BLAST jobs
  • Building own local cluster
  • Submit jobs to NCBI or EBI
    – Long job queuing time

Opportunity for Cloud computing
Windows Azure
Computation Fabric

A bunch of machines in data centers

Fabric Controller

• Owns all data center hardware
• Uses inventory to host services
• Deploys applications to free resources
• Maintains the health of those applications
• Maintains health of hardware
  – If the node goes offline, FC will try to recover it
  – If a failed node can’t be recovered, FC migrates role instances to a new node, A suitable replacement location is found, Existing role instances are notified of change
• Manages the service life cycle starting from bare metal

Highly-available Fabric Controller (FC)
Azure Virtual Machine

At Minimum (Small)
- CPU: 1.5-1.7 GHz x64
- Memory: 1.7GB
- Network: 100+ Mbps
- Local Storage: 500GB

Up to 7 Guest VMs

A Host Virtual Machine
An Optimized Hypervisor

Up to (Extra Large)
- CPU: 8 Cores
- Memory: 14.2 GB
- Local Storage: 2+ TB
Storage Fabric

Blobs: Simple interface for storing named files along with metadata for the file

Drives – Durable NTFS volumes

Tables: entity-based storage
Not relational – entities, which contain a set of properties

Queues: reliable message-based communication
Azure Queue

Communication channel between instances

• Messages in the Queue is reliable and durable
  – 7-day life time

Fault tolerance mechanism

• De-queued message becomes visible again after visibilityTimeout if it is not deleted
  – 2-hour maximum limitation
• Idempotent processing
Suggested Application Model
Using queues for reliable messaging

To scale, add more of either

• Decouple the system
• Absorb the bursts
• Resilient to the instance failure,
• Easy to scale
AzureBLAST

Parallel BLAST engine on Azure
Query-segmentation data-parallel pattern
Follows the suggested application model
  • Web Role + Queue + Worker
Three special considerations
  • Long-Running Job (Batch job)
  • Massive Task parallelism on the elastic platform
  • Large data-set management
Architecture Overview

- Web Portal
- Web Service
- Job registration

Job Management Role:
- Scaling Engine
- Job Scheduler

Global dispatch queue:
- Worker
- Worker
- Worker

- Azure Table
- Database updating Role

NCBI databases
- Azure Blob

- Database
- Temporary data, etc.)
Job Portal Role

ASP.NET program hosted by a web role instance
- Submit jobs
- Track job’s status and logs
Authentication based on Live ID
The accepted job is stored into the Azure table
- Fault tolerance, avoid in-memory states
Job Manager Role

Job scheduling
- FCFS/Priority
  - En-queue the main task of the job
- Maintaining the job state

Scaling engine
- Each job can specify the Min/Max number of instances
- Scale out/in dynamically

Why separate Job Manager from Job Portal?
Worker Role

Each worker instance
- Keep pulling the task message from the dispatch queue
- execute it (run NCBI-BLAST), once done delete the message
- Once receiving the abort signal, terminate the task running

Fault tolerance
- Simply rely on the reliable Azure message

Decoupling via the queue

Global dispatch queue

Blast databases, temporary data, etc.)

Azure Blob
Task Parallelism

Spawning a task
• En-queuing the task message in the \textit{dispatch queue}
Each task has its own
• Output + Cancellation queues
  – Can be redirected to a shared one
Parent task joins the children tasks
• And/Choice/OnException
• Synch. vs. Asynch. (saving cost)
Continuation task
• More fault-tolerant
Exception Handling
• Cancel,
• Retry or Ignore
  – Cost-effective for the embarrassingly parallel tasks
BLAST Tasks

A Simple Split/Join Pattern

BLAST Task

- Invokes the NCBI-BLAST executable directly
- Leveraging multi-core resource by the argument "--a"

Task Granularity

- Large partition $\rightarrow$ load imbalance
- Small partition $\rightarrow$ unnecessary overheads
  - NCBI-BLAST overhead
  - Data transferring overhead.

- **Best practices:**
  - *profiling by test-running and set size to mitigate the overhead*

**VisibilityTimeout** for each BLAST task,

- essentially the estimation of the task running time.
- too small $\rightarrow$ repeated computation;
- too large $\rightarrow$ unnecessary long period of waiting time in case of the instance failure.

- **Best practices:**
  - *estimate the value based on the number of pair-bases in the partition and test-running*
Managing Large Datasets

Bio-sequence databases
- Usually are updated frequently (daily or weekly)
- Most are large (1GB – 10GB)

Data transferring cost
- within cloud is free, otherwise not.

Database updating Role
- Keep tracking the NCBI reference databases
- Refreshes them into Azure blob

Database staging by workers
- Copy database blobs to local disk
- Or mount the snapshot of the blob as Azure Driver

Tips:
- Bio-databases have a high compress ratio (e.g., 28%)
- AzureDrive (beta version) currently is slower
Micro-benchmark

Task size vs. Performance
- benefit of the warm cache effect
- 100 sequences per partition is the best choice

Instance size vs. Performance
- super-linear speedup with larger size worker instances
- primarily due to the memory capability.

Task Size/Instance Size vs. Cost
- Extra-large instance generated the best and the most economical throughput
- Fully utilize the resource you pay for
Scaling

• Static Scaling
  – Instances are allocated statically
  – Databases are staging statically
  – Only measure the job execution time
  – Linear Speedup Up to 64 extra-large instances
    • due to the pleasingly parallel tasks.

• Dynamic Scaling
  – Two consecutive jobs
    • One ask 8 instances, one asked 16 instances
  – Instances allocation is dynamic by scaling engine
  – Database staging is dynamic
  – Be aware of the overhead
  – Don’t scale out synchronously
Blob reading throughput
- is remarkable and scale well
- \( \sim 223 \text{ MB/sec} \) aggregated throughput
- data staging in a lazy manner is feasible.

Blob writing throughput
- is relatively low because of the three independent copies of each blob.
- But it scales well with more instances
A Large Experiment

10 million sequences vs. 10 million sequences
Allocated totally 3776 weighted instances
  • 475 extra-large instances
  • From three datacenters
    – (US South Central, West Europe and North Europe)
8 deployments of AzureBLAST
Two-week Running time
35 Nodes experienced the blob writing failure at the same time.

Network disconnection?

Node failure, a replacement node comes in.

Twice task losing for all nodes during 9 hours at the beginning of April.

System upgrading.
Lessons Learned

Always design with failure in mind

- Failure happens in various forms, at unpredictable time
- Do not make decision on any in-memory state or static machine info
- Avoid these tightly coupled synchronous constructs – such as barrier/lock.

Task Parallelism is a good fit model for the Cloud computing
Decouple critical components via the queue or the table
- For the better fault tolerance, scalability and availability

The better you understand the problem, the more you will save;

- $1 test-run may save you $1000
- Allocate the resource as appropriate.

Fully utilize each resource which you have paid for

- Cache, Cores, Memory and Local Disk

Dynamic scaling is the effective solution to save cost
- But be aware of the overhead of scaling