



Progressive Multiple Biosequence Alignments on the ALiCE Grid

Yong Meng Teo^{1,2}
Yew-Kwong NG¹
Xianbing Wang²

¹*Department of Computer Science, NUS, Singapore 117543*

²*Singapore-MIT Alliance, Singapore 117576*

{teoym, wangxb}@comp.nus.edu.sg


1



Outline

- ALiCE & Motivation
- PMSA Problem
- Distributed PMSA on ALiCE
- Performance Evaluation
- Conclusion


2



ALiCE

- ALiCE is a portable middleware designed for developing and deploying general-purpose grid applications and application programming models.
- ALiCE is designed to meet a number of design goals, such as *flexibility*, *scalability* and *platform independent*.

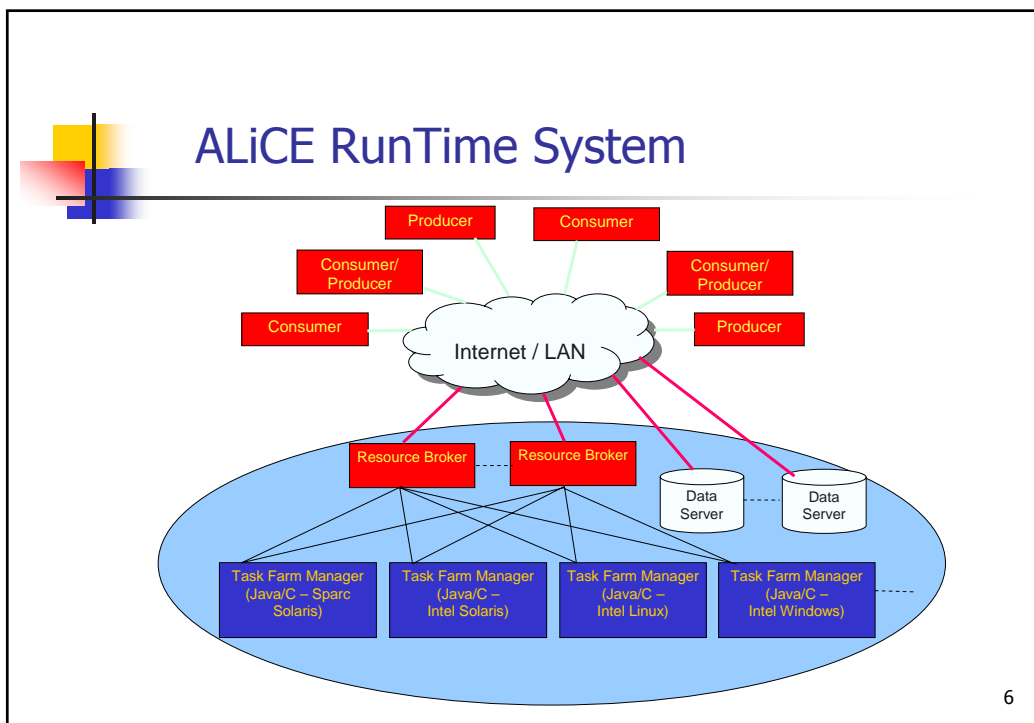
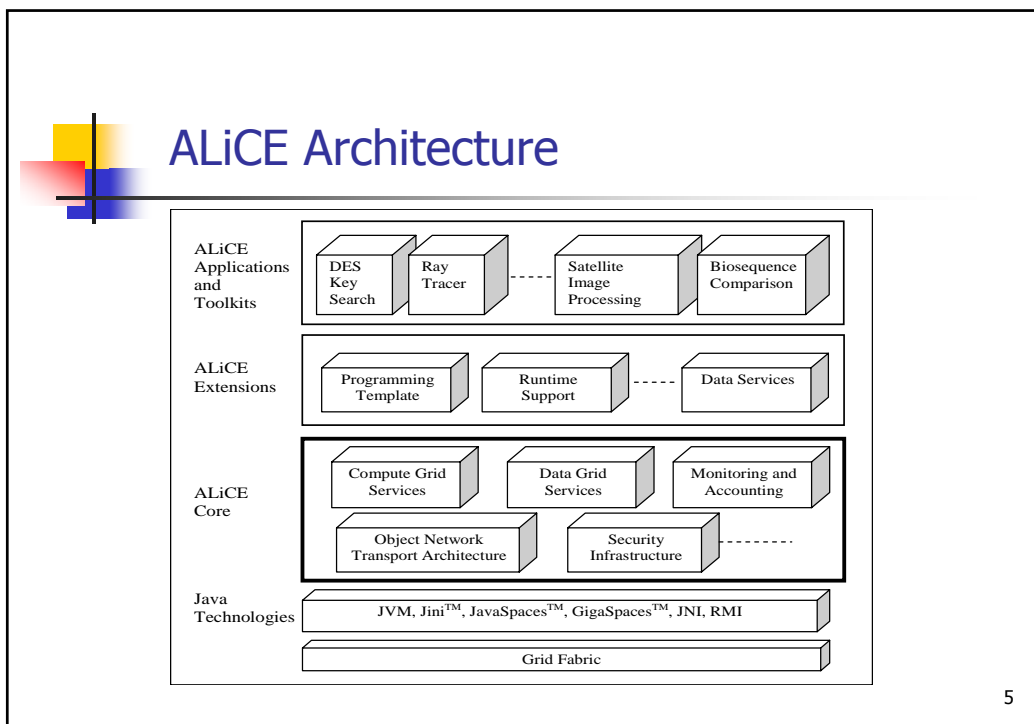
3




Motivation

- Develop an adaptable **programming toolkit** to facilitate the implementation of medium- to large-scale **bioinformatics applications** on grid environments
- Demonstrate the capability of **ALiCE** in deploying **data-intensive** and **compute-intensive** problems for parallel execution on grids

4






ALiCE Programming Template

- ALiCE adopts the *TaskGenerator-ResultCollector* programming model
- *TaskGenerator* runs on a task farm manager machine and allows tasks to be generated for scheduling by the resource broker.
- *Task* runs on a producer machine, and it specifies the parallel execution routine at the producer.
- *Result* models a result object that is returned from the execution of a task.
- *ResultCollector* runs on a consumer machine, and handles user data input for an application and the visualization of results thereafter.

7



<p style="text-align: center;">TaskGenerator Template</p> <pre>import alice.consumer.*; import alice.data.*; public class TASKGEN_CLASSNAME extends TaskGenerator { public TASKGEN_CLASSNAME() {} public void init() { //Place your initialisation code here } /* Main method - entry point */ public void main(String args[]) { //This is where the tasks are generated, usually in a loop //This should be called for each task TASK_CLASSNAME t = new TASK_CLASSNAME(); process(t); // To open a data file, read and write from/to it DataFile f = Data.openFile("file_name",this); READ_BUFF = f.read(POSITION, LENGTH); f.write(WRITE_BUFF, POSITION, LENGTH); // To send/receive an object OBJECT_CLASSNAME obj = new OBJECT_CLASSNAME(); sendObject(obj, "snd_str_id"); OBJECT_CLASSNAME rcvObj = (OBJECT_CLASSNAME) requestObject("rcv_str_id"); // To receive a string message from the result collector: String msg = getStringMessage(); } }</pre>	<p style="text-align: center;">Task Template</p> <pre>import alice.consumer.*; import java.io.*; public class TASK_CLASSNAME extends Task { // Place variables here public TASK_CLASSNAME () { } public Object execute () { // This is where you do your computations. The results can be any kind of // objects // You can generate and send a new task to be produced O_TASK_CLASSNAME t = new O_TASK_CLASSNAME(); process(t); // To open a data file, read and write from/to it DataFile f = Data.openFile("file_name",this); READ_BUFF = f.read(POSITION, LENGTH); f.write(WRITE_BUFF, POSITION, LENGTH); // To send/receive an object OBJECT_CLASSNAME obj = new OBJECT_CLASSNAME(); sendObject(obj, "snd_str_id"); OBJECT_CLASSNAME rcvObj = (OBJECT_CLASSNAME) requestObject("rcv_str_id"); } }</pre>
<p style="text-align: center;">Result Template</p> <pre>import java.io.*; public class MyResult implements Serializable { public DATA_TYPE var; public MyResult() { var=NULL; } }</pre>	<p style="text-align: center;">ResultCollector Template</p> <pre>import alice.result.*; public class RESCOL_CLASSNAME extends ResultCollector { // Place Variables Here public RESCOL_CLASSNAME () { } public void collect() { // Place here the result collection and processing code to obtain // number of results ready call int resReady = getResultsNoReady(); // To get a new result call RES_CLASSNAME res = (RES_CLASSNAME)collectResult(); } }</pre>

8

Progressive Multiple Sequence Alignment

- The simultaneous alignment of multiple sequences is now an essential methodology in molecular biology.
- to find diagnostic patterns to characterize protein families
- to detect or demonstrate homology between new sequences and existing families of sequences
- to help predict the secondary and tertiary structures of newly sequenced genes
- to suggest special primers for Polymerase Chain Reaction, etc.

9

The Three Successive Stages

Sequence Databases

Pairwise Comparison

-	-	-	-
0.25	-	-	-
0.78	0.47	-	-
0.34	0.19	0.57	-

Profiles Alignment

```

TGCAAGTTAACCAATTTTATGGAACCAAG
TTTTTTTGGCTGGTCAAGGCTTTATTCC
TTTTTTTAAATTTTAAAGCTTTTITTT
TTTTTTTTTAGAAAGAGGTCAAAGTACCT
TTTTGCTTCAGTGCATTGGCCATTTTAT
TTGGCTCTGACTTCTTTGGAAGAGGAGA
TTTTTTTTTCCCTAAATGGTAGGATT
TTTTTTAAATTTCTGCAGTTTATTCTT
                    
```

Guide Tree

```

      A 0.23
     /  \
    Hbb_Human
    Hbb_Horse
   /  \
  C 0.15
 /    \
B 0.25
 /  \
Hba_Human
Hba_Horse
/
D 0.32
/
Myg_PhycA
                    
```

10

Stage 1: Pairwise Comparison

- **Objective** – To derive the *similarities* between each pair of sequences in the database
- Similarity Score Function of biosequences (s, t)

$$sim(s[1..i], t[1..j]) = \max \begin{cases} sim(s[1..i], t[1..j-1]) - 2 \\ sim(s[1..i-1], t[1..j-1]) + p(i, j) \\ sim(s[1..i-1], t[1..j]) - 2 \end{cases}$$

11

Stage 1: Pairwise Comparison-II

- Similarity Scores Matrix example (after comparison)

Hbb_Human	1	$\begin{pmatrix} -- & & & & & \\ & \mathbf{0.23} & -- & & & \\ & 0.41 & 0.40 & -- & & \\ & 0.41 & 0.41 & \mathbf{0.25} & -- & \\ & 0.83 & 0.33 & 0.87 & 0.27 & -- \end{pmatrix}$				
Hbb_Horse	2					
Hba_Human	3					
Hba_Horse	4					
Myg_Phyca	5					
		1	2	3	4	5

12

Stage 2: Guide Tree Construction

- **Objective** – To produce the sequence in which progressive alignment will take place later, so that the more closely related sequences and profiles are aligned first.

13

Stage 2: Guide Tree Construction-II

- Build tree based on best (lowest) similarity score values in matrix at each stage.

```
graph LR; D((D 0.32)) --- C((C 0.15)); D --- Myg_Phyca[Myg_Phyca]; C --- A((A 0.23)); C --- B((B 0.25)); A --- Hbb_Human[Hbb_Human]; A --- Hbb_Horse[Hbb_Horse]; B --- Hba_Human[Hba_Human]; B --- Hba_Horse[Hba_Horse];
```

14



Stage 3: Progressive Alignment

- **Objective** – To cluster the sequence profiles according to the guide tree branches to produce the final alignment of all sequences.

15



PMSA Algorithm

```

Progressive_Multiple_Sequence_Alignment ( $D$ )
1    $n = |D|$ 
2    $M = (n \times n)$  distance matrix

PC {
3   for all  $s \in D$  do
4     for all  $t \in D - \{s\}$  do
5        $M[\text{index}(s), \text{index}(t)] \leftarrow \text{sim}(s, t)$ 
6
7    $T =$  guide tree
8    $T \leftarrow \phi$ 
9    $G =$  number of sequence groups remaining
10   $G \leftarrow D$ 
11  while  $|G| > 1$  do
12     $v \leftarrow$  minimum value in  $M$ 
13     $c \leftarrow$  sequence group formed by  $s$  and  $t$ , where  $M[\text{index}(s), \text{index}(t)] = v$ 
14    Recompute values in  $M$  that involves  $s$  and  $t$ 
15     $G \leftarrow G - \{s, t\}$ 
16     $G \leftarrow G \cup \{c\}$ 
17    Add  $c$  to  $T$ , and add edges from  $c$  to  $s$  and  $t$  respectively in  $T$ 
18
19   $d \leftarrow$  depth of  $T$ 
20  while  $d > 0$  do
21    for all  $s \in T, t \in T, \text{depth}(s) = d, \text{depth}(t) = d, \text{parent}(s) = \text{parent}(t)$  do
22       $(\text{align}_s, \text{align}_t) \leftarrow$  Align  $s$  and  $t$ 
23       $s \leftarrow \text{align}_s$ 
24       $t \leftarrow \text{align}_t$ 
25       $\text{parent}(s) \leftarrow \text{merge}(\text{align}_s, \text{align}_t)$  // creates alignment profile
26       $d \leftarrow d - 1$ 
27
28  return  $T - \{c \mid c \text{ is not a leaf of } T\}$ 

```

16

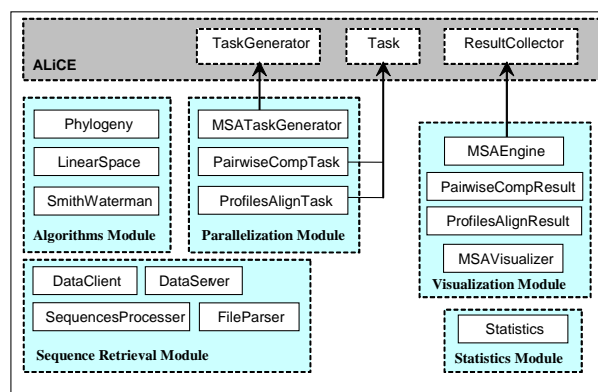
Parallelization of MSA

Given a database with n sequences,

- Stage 1: Each task compares a sequence to d other sequences in the database.
Total no. of tasks $\approx n(n-1) / 2d$
- Stage 2: *No parallelization*
- Stage 3: Each task performs the alignment of a pair of sequence profiles. All alignments at the same depth in the guide tree can proceed in parallel, since they are mutually independent tasks. **Total no. of tasks = No. of profile alignments**

17

Implementation in ALiCE



18



PMSA Implementation

- **Algorithms** encompasses the bioinformatics algorithms used in PMSA.
- **Parallelization** handles the distribution and execution of the tasks in the pairwise comparison and profiles alignment stages.
- **Sequence Retrieval** retrieves the necessary sequences for computation.
- **Statistics** compile meta-information.
- **Visualization** collects and presents the execution results to the user via a GUI.

19



Performance Evaluation

- **Experimental Environment**
 - **Grid I (Homogeneous cluster grid)**
 - 64 Intel Xeon 1.4GHz dual processors nodes, each with 1GB RAM, connected via a *Myrinet* switch
 - **Grid II (Heterogeneous cluster grid)**
 - 26 Intel Xeon 1.4GHz dual processors nodes, each with 1GB RAM, connected via a *Myrinet* switch
 - 16 Pentium II 400MHz, each with 256MB RAM
 - 8 Pentium III 866MHz, each with 256MB RAM
 - *Fast Ethernet* connection between the 2 grid segments

20

Performance Evaluation-II

- Study how performance scales with computational power
- Sequence database centralized on NFS
- Database size = 8,000 sequences
(~ 10MB)

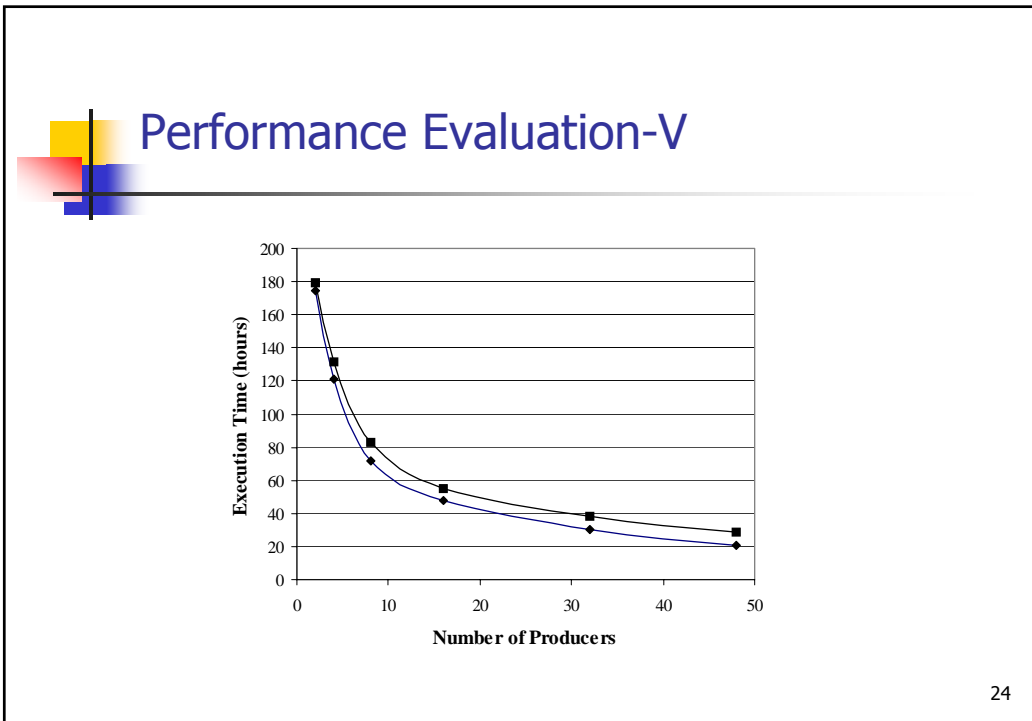
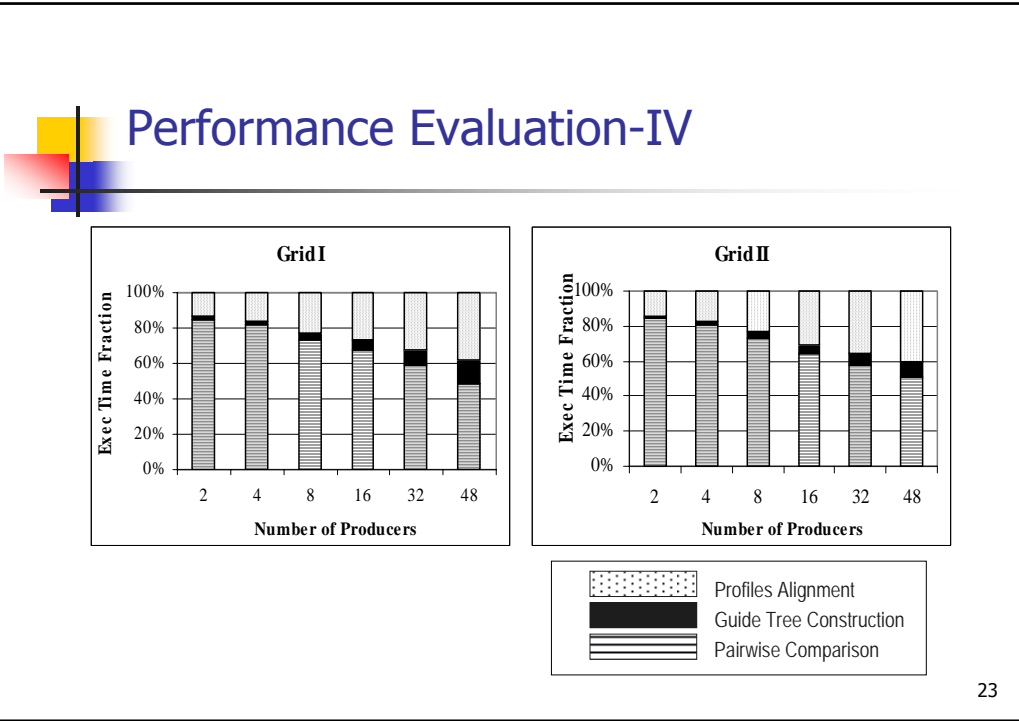
21

Performance Evaluation-III

- Sequential execution time = 237.6 hours

#Producers	Execution Time (hours)							
	Grid I				Grid II			
	PC	GT	PA	Total	PC	GT	PA	Total
2	148.1	2.6	23.5	174.2	151.9	2.6	25.1	179.6
4	98.6	2.6	19.7	120.9	106.1	2.6	22.8	131.5
8	52.2	2.6	16.6	71.4	61.1	2.6	19.5	83.2
16	32.4	2.6	12.9	47.9	35.6	2.6	17.1	55.3
32	17.8	2.6	9.8	30.2	22.1	2.6	13.8	38.5
48	10.2	2.6	8	20.8	14.7	2.6	11.7	29

22





Conclusion and Future Works

- Parallelized PMSA problem on the ALiCE grid.
- Study the performance scalability of grid-based PMSA.
- The next approach is to develop a grid programming model for life sciences applications.
- Facilitates the deployment of complicated bioinformatics problems on the grid without re-implementing commonly used component algorithms that can be very tedious to parallelize.

25



Q & A

26