Enabling Grids for E-sciencE

Grid Optimisation

Heinz Stockinger
Swiss Institute of Bioinformatics

www.eu-egee.org

Information Society and Media

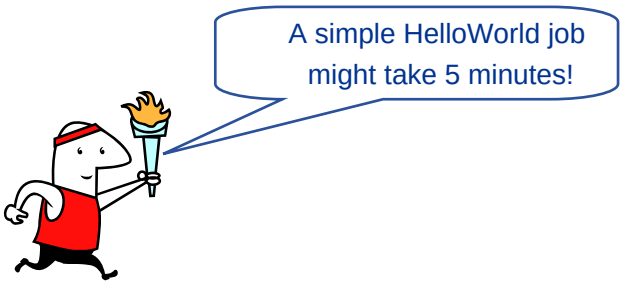
Enabling Grids for E-sciencE

Overview

- Motivation & terminology
- Performance of Grid job submission
- How to speed up applications
- Performance example
- Message Passing Interface and the Grid
- Data and protocol related issues

Motivation: Performance

- **Why are Grid job so “slow”?**



- **Performance does not necessarily mean that one gets immediate response**

Terminology

- **High performance computing:**

Optimise a **single application**.
Execute it as quickly as possible.

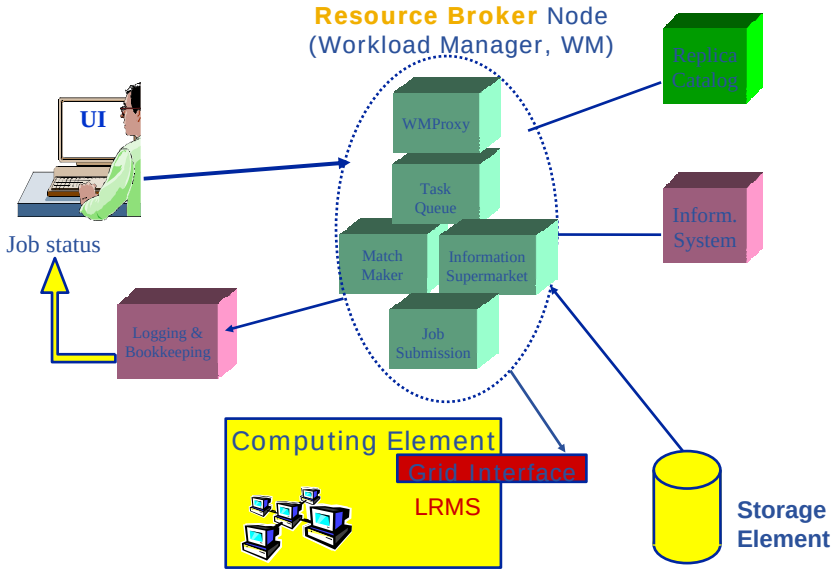
- **High throughput computing:**

Optimise a **set of concurrent applications**.
Give certain time to each application.

Overview

- Motivation & Terminology
- **Performance of Grid job submission**
- How to speed up applications
- Performance example
- Message Passing Interface and the Grid
- Data and protocol related issues

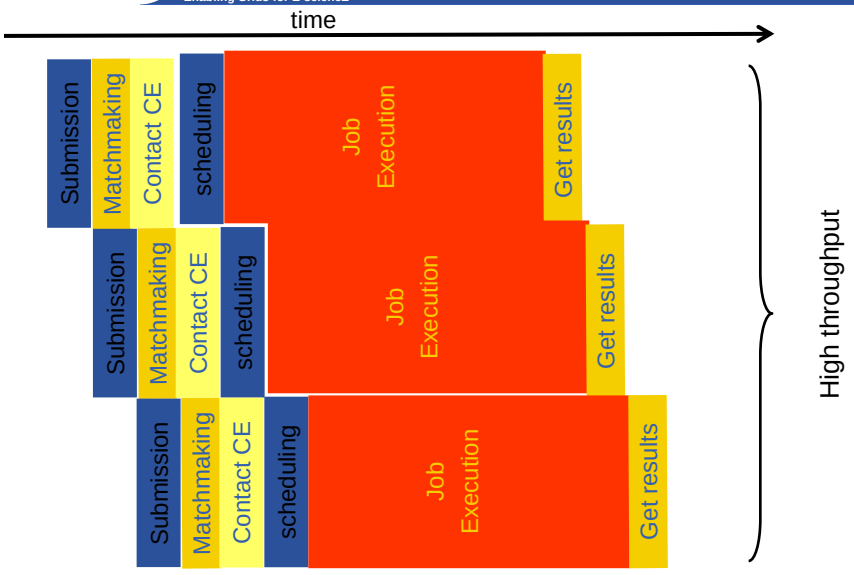
Complex Interactions



Overall WMS Performance

- **Sum of**
  - **Connection** of client to broker over WAN
    - RTT is between 30 and 150 ms
  - Resource selection and **matchmaking** at broker
    - Contacting Information System
  - **Sending job wrapper** to Computing Element
    - Includes secure interaction via GSI
  - Computing Element **passes job to LRMS**
    - Includes secure interaction via SSH
  - LRMS **selects a worker node**
    - Job is sent to worker node
  - **Logging and bookkeeping** needs to be updated
  - **Store output** at Resource Broker node

Pipelining



More on Grid Job Performance

Enabling Grids for E-sciencE

Submission
Matchmaking
Contact CE
scheduling
Job Execution
Get results

- **The Grid is not optimised for small jobs**
  - Every job will have this additional latency of about 3-5 min
  - Interesting for “long” running jobs (e.g. more than 30 min)
- **Sometimes scheduling at the LRMS takes long**
  - LRMS is a batch system (not interactive!)
  - Many jobs might be already in the waiting queue
  - Some systems (queues) are set up in a way that they only allow 1 or 2 job(s) per processor
    - Performance for “single” users
  - In EGEE: “short deadline jobs”
- **A Grid has heterogeneous hardware resources**
  - Some clusters have faster processors than others

The Grid and Processor Speed

Enabling Grids for E-sciencE

- **Faster processor = better performance?**
- **By default, the WMS does not select “fastest” processors but sites with many processors and short queues**
  - *Try it out!*
  - One can give hints to the system by specifying job requirements for certain processor speed
  - However, one might need to “wait” for fast processors to become free and the job might be done by “slower” processors in the meantime

Overview

- Motivation & Terminology
- Performance of Grid job submission
- **How to speed up applications**
- Performance example
- Message Passing Interface and the Grid
- Data and protocol related issues

Grid: High Throughput Computing

- **Grids are often not very good at running a small task very fast**
- **However, due to lots of computing power and storage, it can speed up computing intensive, big task:**
  - By parallelising the workload
  - Or by simply offering more computing power
- **Automatic parallelisation not really achieved**

Choose you application carefully that you want to run on the Grid.
Check for potential parallelism.

Which Type of Application?

Enabling Grids for E-sciencE

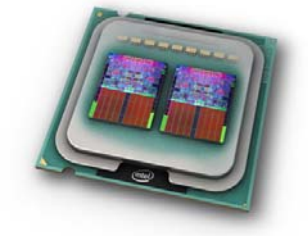
Any ideas?

How to make things faster?

Enabling Grids for E-sciencE

- **When you design a program newly, think about work flow scenarios that can be executed concurrently**
  - Overlapping computation and I/O is only a small step
- **Check if input data set can be divided into smaller chunks**
  - If yes, branch off, run in parallel and merge in the end
- **Legacy applications:**
  - We often cannot change existing code
    - Detector simulation code should not be touched by non-experts
  - If possible, write job wrappers

How about that?

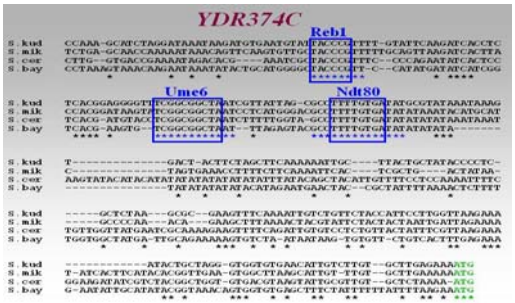


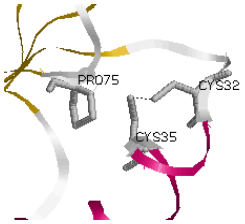
Overview

- Motivation & Terminology
- Performance of Grid job submission
- How to speed up applications
- **Performance example**
- Message Passing Interface and the Grid
- Data and protocol related issues

Bioinformatics Example

- **Biological sequence analysis**

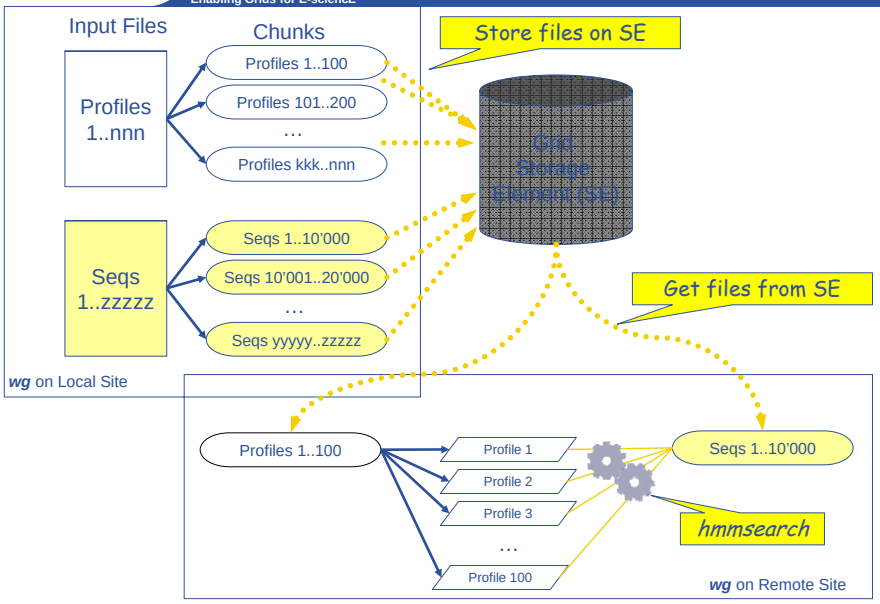




- **CPU intensive, embarrassingly parallel problem**
  - Takes to databases as input
  - Input data is split into chunks which can be processed in parallel
  - Ideal for the Grid

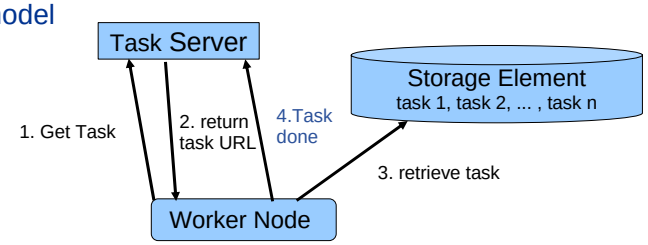
The EMBRACE project is funded by the European Commission within its FP6 Programme, under the thematic area "Life sciences, genomics and biotechnology for health,"contract number LHSG-CT-2004-512092.

hmmsearch profile sequence

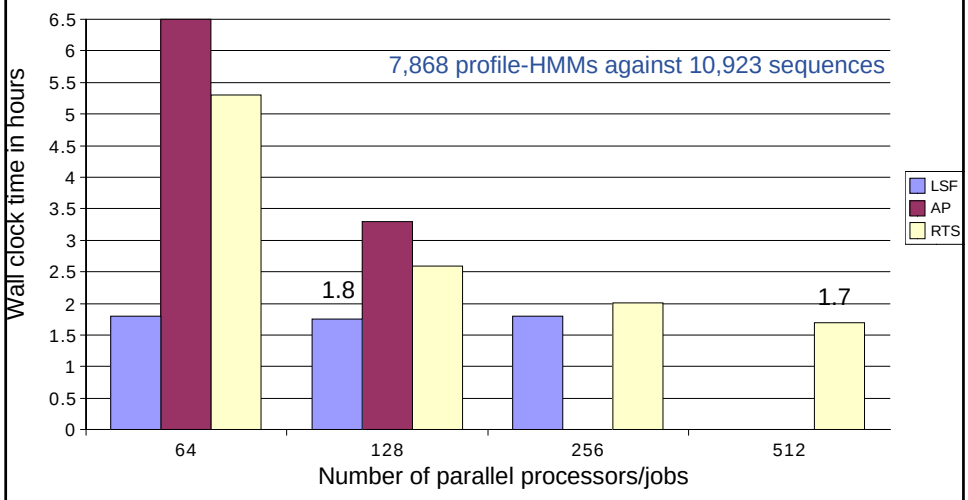


Sequence Analysis (cont.)

- **A-priory algorithm (AP-algorithm)**
  - Decide the task to be computed a-priory based on the number of input data chunks
  - Push model
- **Run time sensitive algorithm (RTS-algorithm)**
  - Decide for a certain number of CPUs
  - Introduce a Task Server that assigns tasks to CPUs
  - Run time based task assignment
  - Pull model

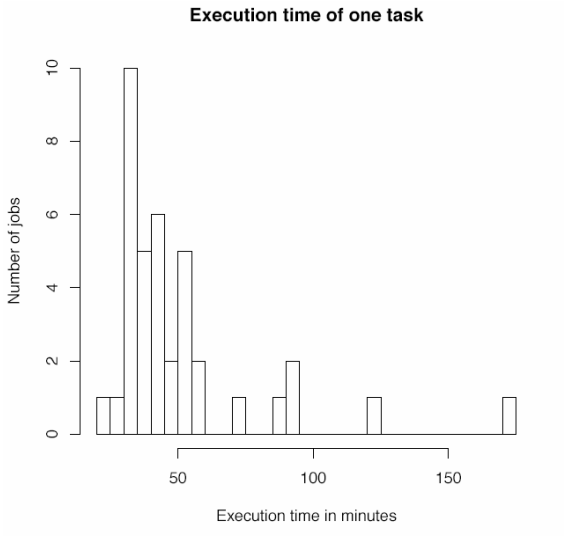


Running a Biological Benchmark

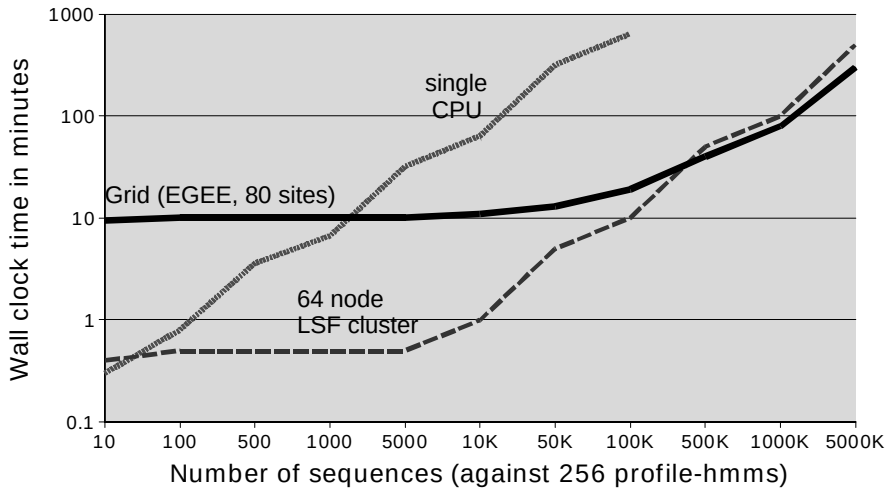


hmmsearch is a standard bioinformatics package for sequence analysis

RTS Algorithm



Execution Time of RTS-Algo.

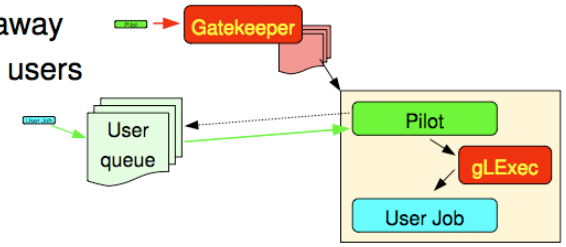


How can this be applied to HEP?

- **What we learned in the biology application can also be applied to HEP**
- **Often called "Pilot jobs"**

Requiring pilot jobs to use gLExec closes the loop

- Sites know who is running
- Can turn away unwanted users



(based on slide by Igor Sfiligoi)

Overview

- Motivation & Terminology
- Performance of Grid job submission
- How to speed up applications
- Performance example
- **Message Passing Interface and the Grid**
- Data and protocol related issues

High Performance Computing

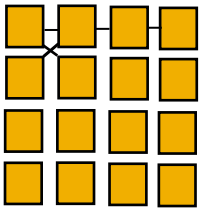
Enabling Grids for E-sciencE

- **Massive parallelism**
  - One single application runs on several processors that have a high-speed interconnect (network)
  - Traditionally done on massive parallel machines (Cray X, IBM SPx, etc.)
    - Now clusters are used more and more
  - Standards such as **MPI** (Message Passing Interface) are used
    - **Inter-process communication** is required
  - Classical domain: High Performance Computing (HPC)
- **Embarrassing parallelism**
  - Typically, one process initiates the computation and potentially also assigns the workload
    - See bioinformatics example
    - Monte Carlo simulation in HEP
  - No communication between processors

Message Passing Interface

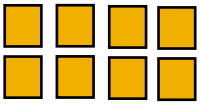
Enabling Grids for E-sciencE

- **De facto standard for HPC requiring message passing**
- **Allows for writing different kinds of (massively) parallel applications**
  - Singe Program Multiple Data (SPMD) approach
  - Execute the same executable on all processors involved in the computation
- **Used in computational sciences**
  - "No" usage in High Energy Physics
  - Commonly used in bioinformatics, engineering etc.



MPI Example

```
#include <stdio.h>
#include <mpi.h>

void main(int argc, char **argv)
{
      int node;
      MPI_Init(&argc,&argv);
      MPI_Comm_rank(MPI_COMM_WORLD, &node);
      printf("Hello World from Node %d¥n",node);
      MPI_Finalize();
}
```

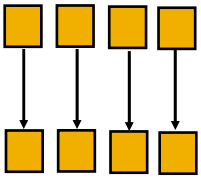


Output:

```
Hello World from Node 2
Hello World from Node 0
Hello World from Node 4
Hello World from Node 9
Hello World from Node 3
Hello World from Node 8
Hello World from Node 7
Hello World from Node 1
Hello World from Node 6
Hello World from Node 5
```


MPI send-recv Example

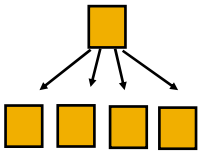
```
int nprocs, myrank, dest, source,  merror, tag=1, itag=2;
char inmsg[128], outmsg[128]  ="Hello from root to ";
char newmsg[128];
static char strd[128];
MPI_Status Status;
int i, m;
size_t msglen;

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

if (myrank == 0) {
   for  (i = 1; i < nprocs; i++) {
      dest  = i;
      m  = sprintf(strd, "%d", dest);
      strcpy(newmsg,  outmsg);
      strcat(newmsg,strd);
      msglen  = strlen(newmsg);
      merror  = MPI_Send(&msglen, 1, MPI_INT, dest, itag,  MPI_COMM_WORLD);
      merror  = MPI_Send(&newmsg, msglen, MPI_CHAR, dest,  tag, MPI_COMM_WORLD);
   }
}
else {
     source  = 0;
     merror  = MPI_Recv(&msglen, 1, MPI_INT, source, itag, MPI_COMM_WORLD,&Status);
    merror  = MPI_Recv(&inmsg, msglen, MPI_CHAR, source,  tag, MPI_COMM_WORLD, &Status);
    printf("Processor  %d received \"%s\" from Processor 0\n",  myrank, inmsg);
}
```



Source: http://www.ats.ucla.edu/rct/hpc/parallel_computing/mpi-intro.htm

Example With Broadcast

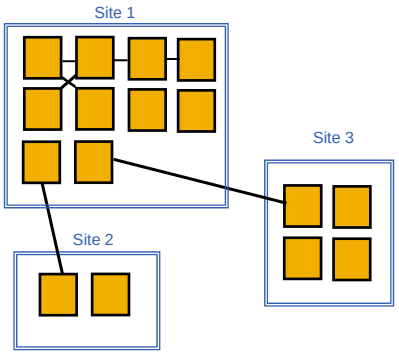
```
int count, pool_size, my_rank, my_name_length, i_am_the_master = FALSE;
char my_name[BUFSIZ], master_name[BUFSIZ], send_buffer[BUFSIZ], recv_buffer[BUFSIZ];
MPI_Status status;

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &pool_size);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Get_processor_name(my_name, &my_name_length);

if (my_rank == MASTER_RANK) {
  i_am_the_master = TRUE; strcpy (master_name, my_name);
}

MPI_Bcast(master_name, BUFSIZ, MPI_CHAR, MASTER_RANK, MPI_COMM_WORLD);

if (i_am_the_master)
  for (count = 1; count < pool_size; count++) {
    MPI_Recv (recv_buffer, BUFSIZ, MPI_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    printf ("%s\n", recv_buffer);
  }
else {
  sprintf(send_buffer, "hello %s, greetings from %s, rank = %d",
          master_name, my_name, my_rank);
  MPI_Send (send_buffer, strlen(send_buffer) + 1, MPI_CHAR,
            MASTER_RANK, 0, MPI_COMM_WORLD);
}
MPI_Finalize();
```



Source http://beige.ucs.indiana.edu/I590/node62.html

MPI and the Grid

- **Massive parallel applications typically require fast networks with similar latencies between the processors**
- **Grids and wide-area networks are very heterogeneous**
- **MPI is supported by EGEE**
  - On single clusters only
- **Often, it makes no sense to run MPI code across sites**
  - Latency issue
  - Worker node might sit behind firewall



Overview

- Motivation & Terminology
- Performance of Grid job submission
- How to speed up applications
- Performance example
- Message Passing Interface and the Grid
- **Data and protocol related issues**

Data Access Optimisation

- **A Data Grid typically provides large volumes of replicated data**
- **Some data items are more frequently accessed**
- **Goal: access to "close" data**
  - With smallest network latency
- **EGEE currently does not have a "data access" monitor which automatically creates/deletes data items**
  - There is a well studied theory about that in database research
  - Similar to the caching mechanisms

Data Transfer

- **An important part of a Data Grid is to deliver files efficiently and securely**
  - Easy, we learned about GridFTP and parallel streams ☺
  - However, for whom do we optimise?
    - For a single user?
    - For a user community (experiment?)

Data Transfer Tuning

- **Using parallel streams is only one way to optimise network bandwidth utilisation**
  - However, 8-16 streams are often good enough
- **Need to optimise TCP buffer size**

TCP buffer size = RTT x speed of bottleneck link

Protocol Issues

- **Services in a Grid need to interact with each other**
- **Grid or Web services typically use SOAP**
- **However, SOAP is not the "fastest protocol"**
  - Good for simple client-server interactions
  - Bad for applications with lot's of communication and/or many large data types to be passed on
- **Other high performance protocols:**
  - xrootd protocol for accessing data in ROOT
  - LFC's protocol to deliver replica locations

Web services using SOAP are great for interoperability.
For mission critical applications they can be performance killers.

Conclusion

- **Grid systems are typically very complex**
- **Making things "work" is hard**
- **Makings things "run fast" is even harder**
  - However, important to keep in!
- **Try to extract parallelism of your application**