

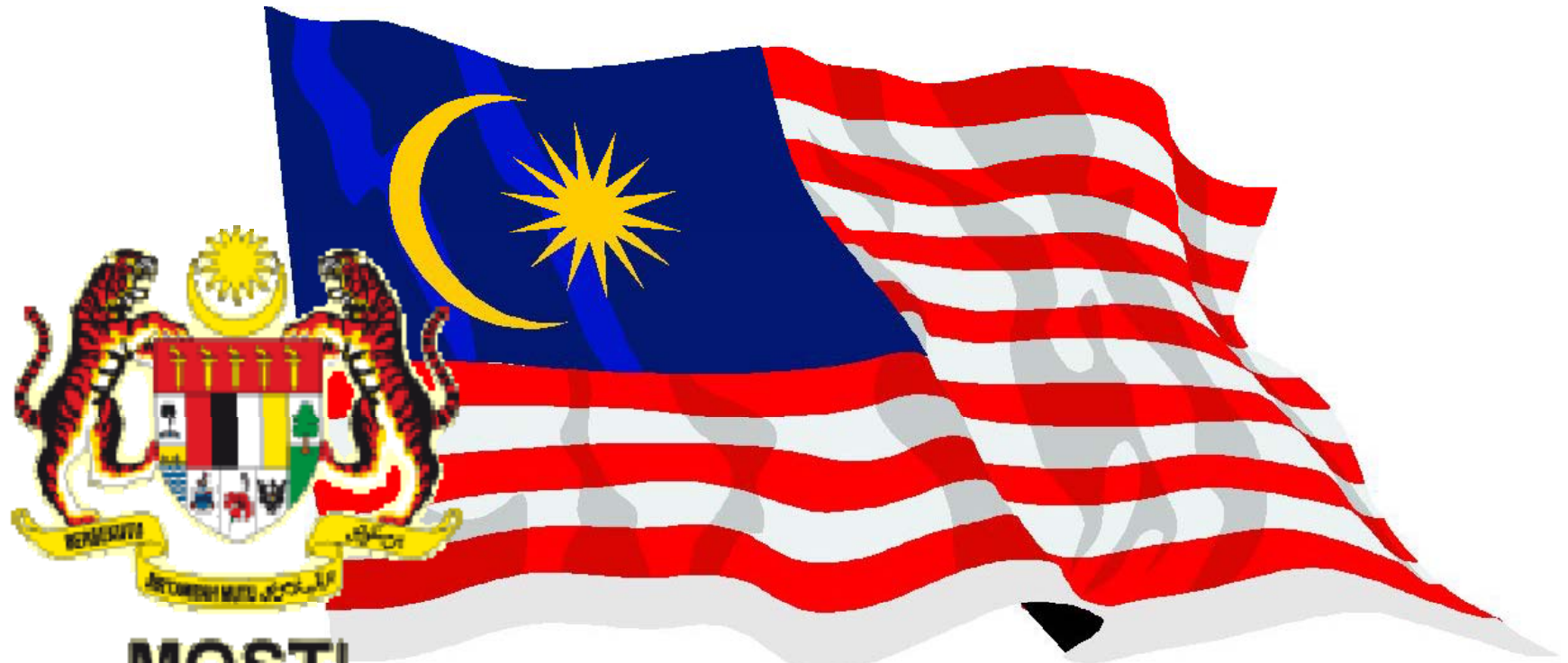


Innovation for Life

Execution Time Prediction of Imperative Paradigm Tasks for Grid Scheduling Optimization

Lai Weng Kin, Yap Yee Jiun,
Maleeha Kiran, ***Lim Mei Kuan**

13 February 2008



MOSTI

Ministry of Science, Technology
and Innovation Malaysia



**Driving INNOVATION
Towards a Knowledge Nation**





brings
SUPER COMPUTING POWER
to all sectors across the nation



Industrial Sector



**Government
Sector**



Agriculture Sector



**International
Community**



Research Community



Financial Sector



Public Sector



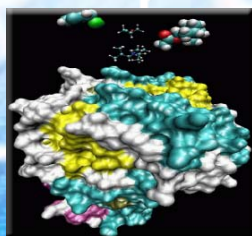
**Business & Enterprise
Sector**

Collaborative Platform across the sectors

aggregate and
improve
efficiency of ICT

reduce
Total Cost of
Ownership (TCO)

enable
Knowledge
sharing



Bio-
researchers
discover
new drugs



Structural
designers
simulate stress
environment



Animation
industry
(Geng the
Movie)



Doctors
predict
cancer
growth



Bankers
perform
financial &
risk
analysis



Command
Center
for natural
disaster-
alerts public



Cyber
police
monitors
national
borders

System View

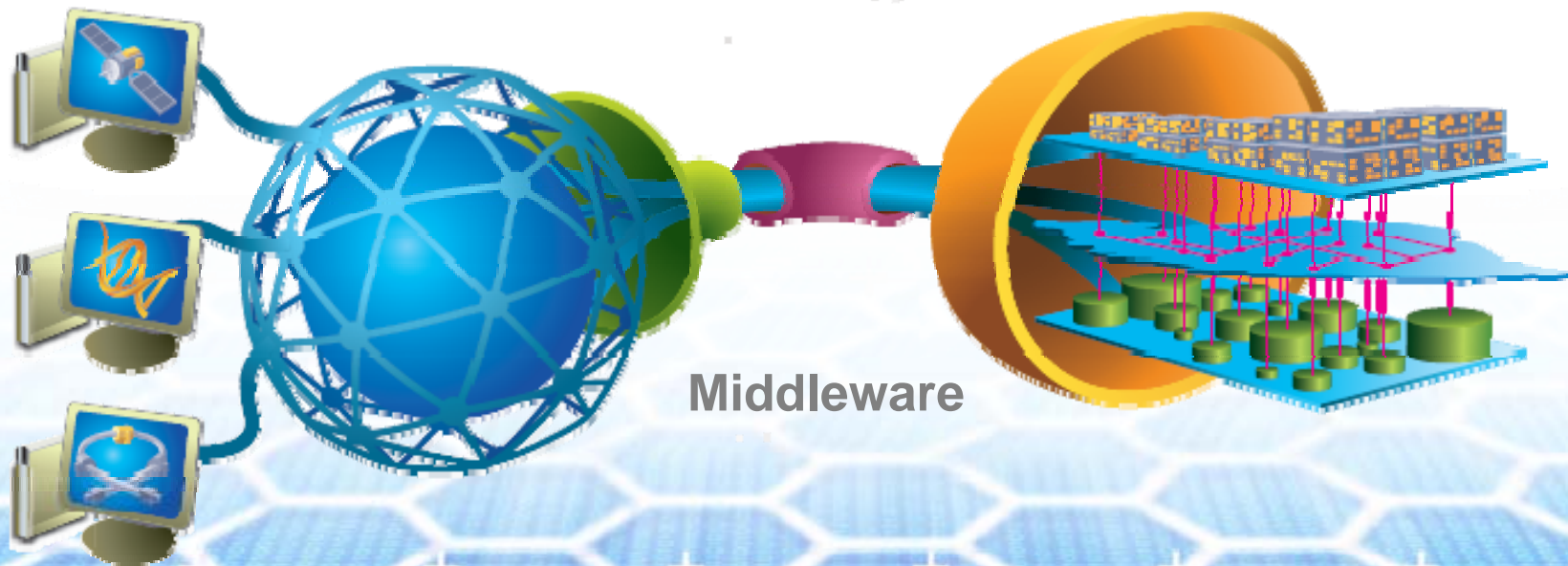


KnowledgeGRID Malaysia website hosts the grid portal which allow user to utilize Grid System

KnowledgeGRID

Unified
Gateway

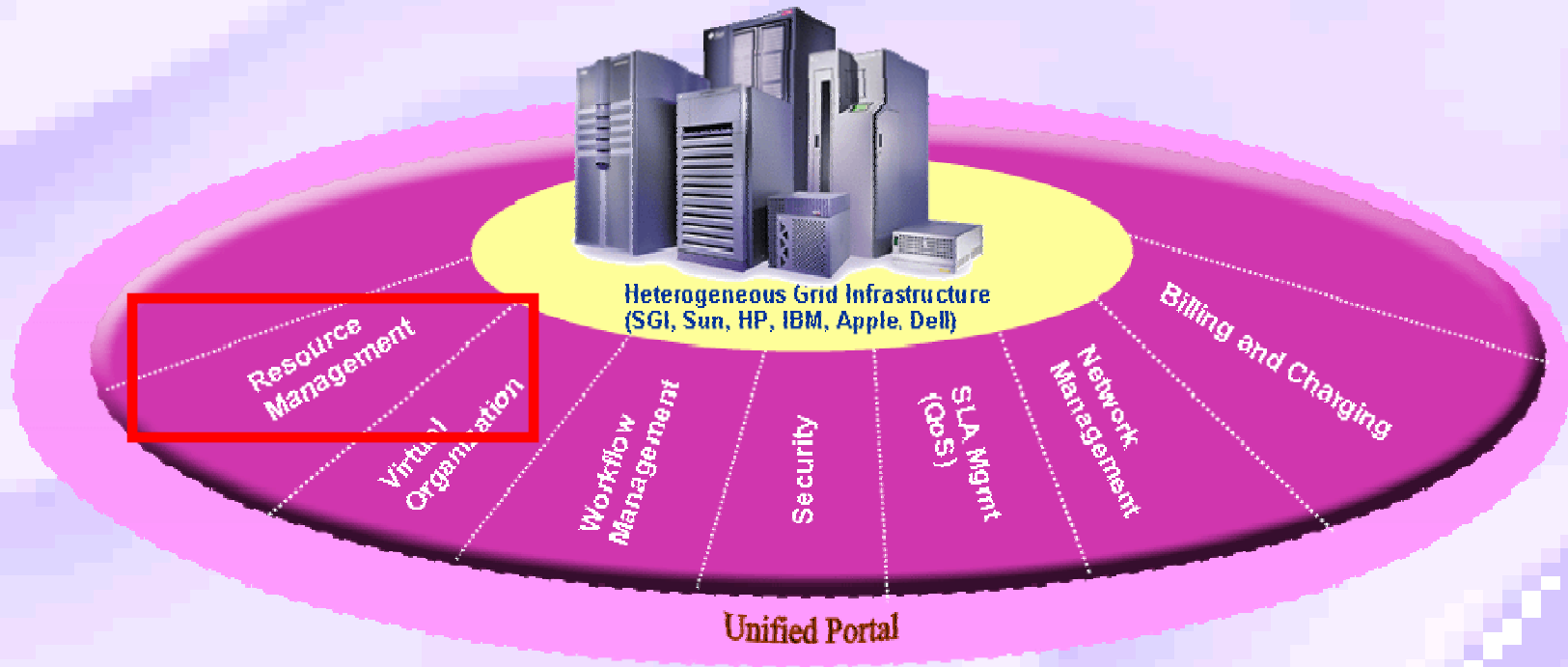
Computing Hardware &
Applications



Middleware

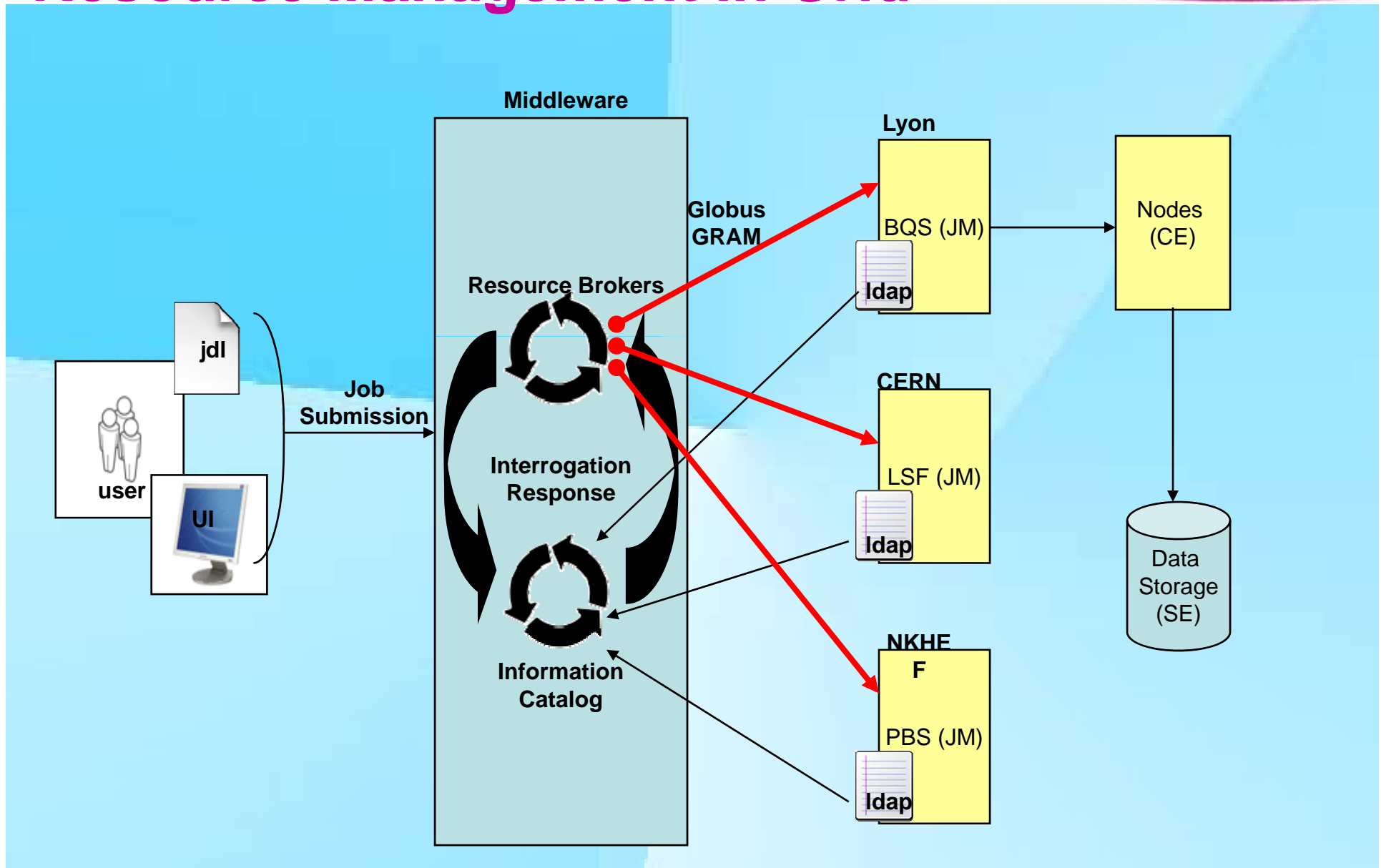
Grid Computing is a form of distributed system where computing resources are shared across networks

Prediction Module – Area in Grid Computing



Our work fits in the area of resource management.
Prediction module that estimates the execution time of jobs to assist in grid scheduling.

Resource Management in Grid



Resource Management in Grid

- Upon submission, user specify the resources needed to run job using Job Description Language (JDL).
- Resource broker finds resources that fits user's requirements through the information catalogue, negotiate with grid-enabled resources, schedule tasks to specified resources and deploy the application.
- Finally, the resource broker gathers results and return them to users.

Analogy of Problem Statement



2 Burgers



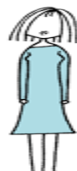
5 Burgers



50 Burgers



1 Burger



2 Burgers



15 Burgers



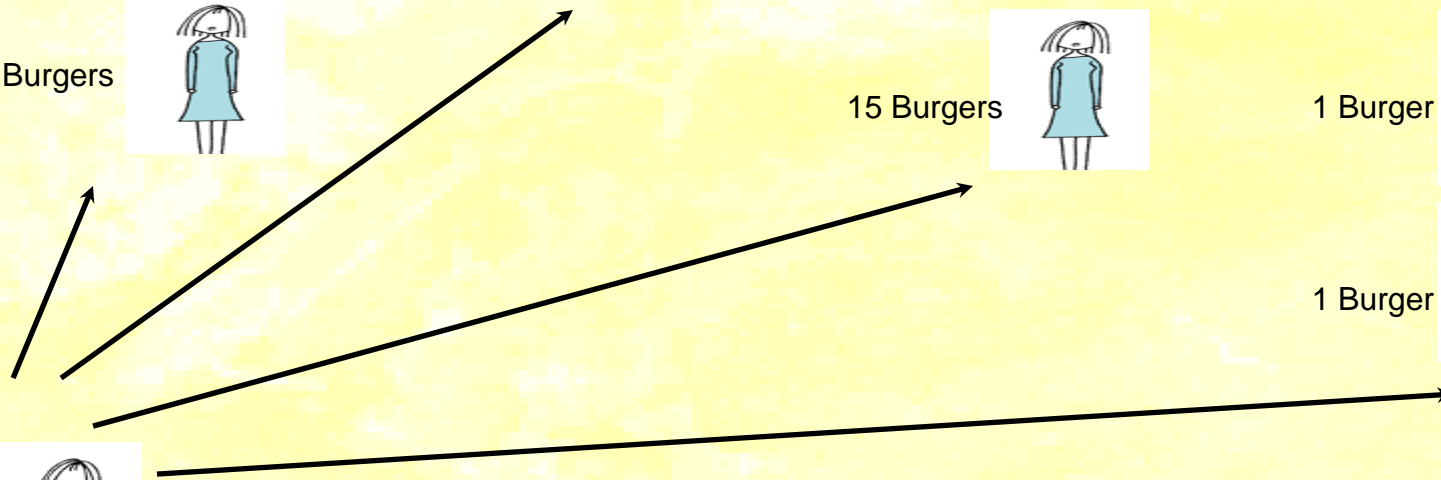
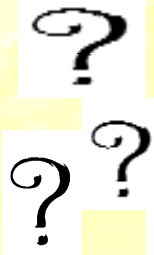
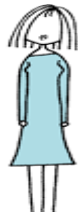
1 Burger



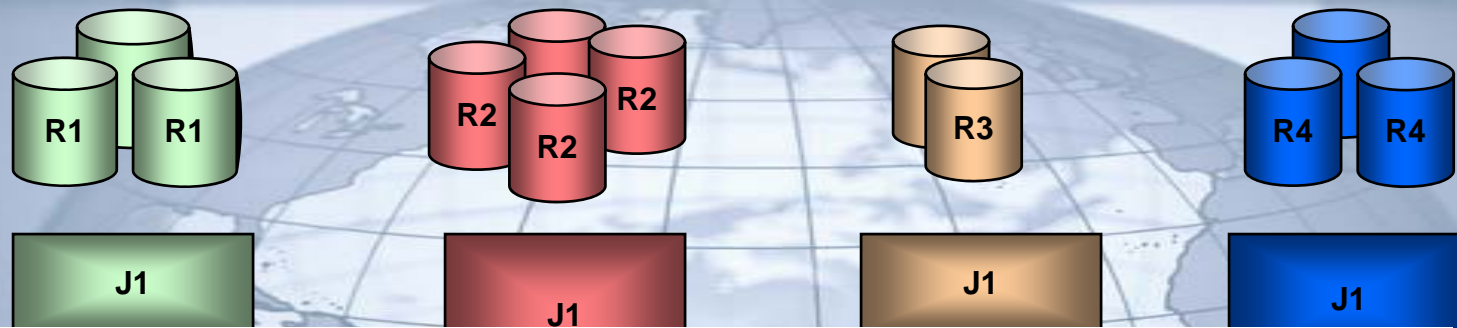
1 Burger



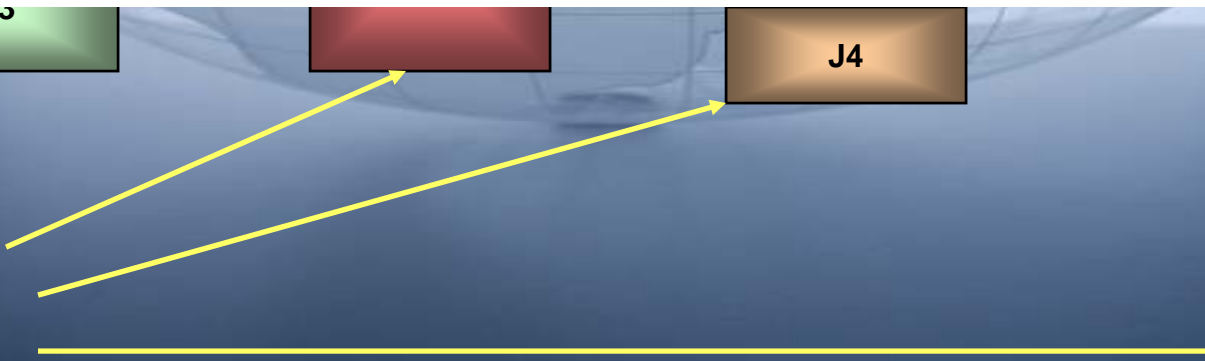
3 Burger



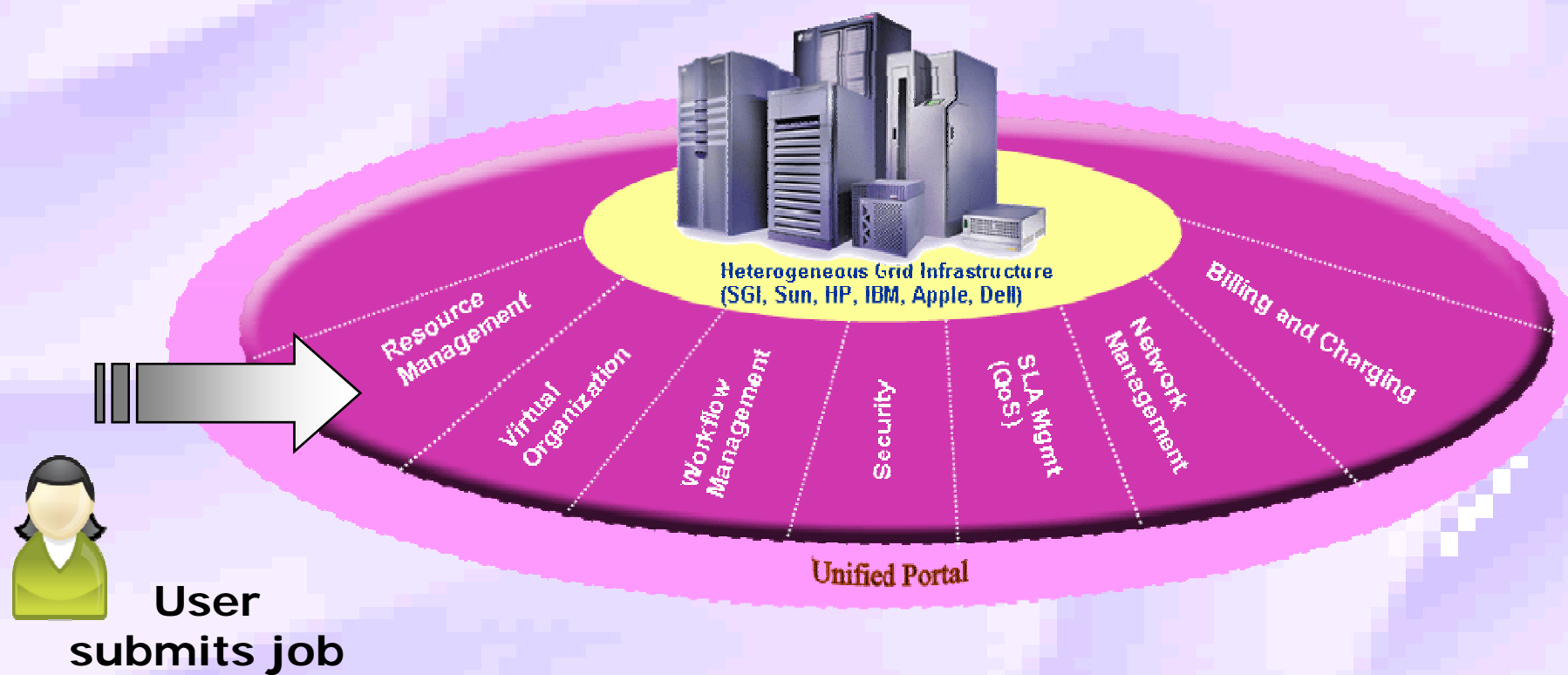
Problem Statement



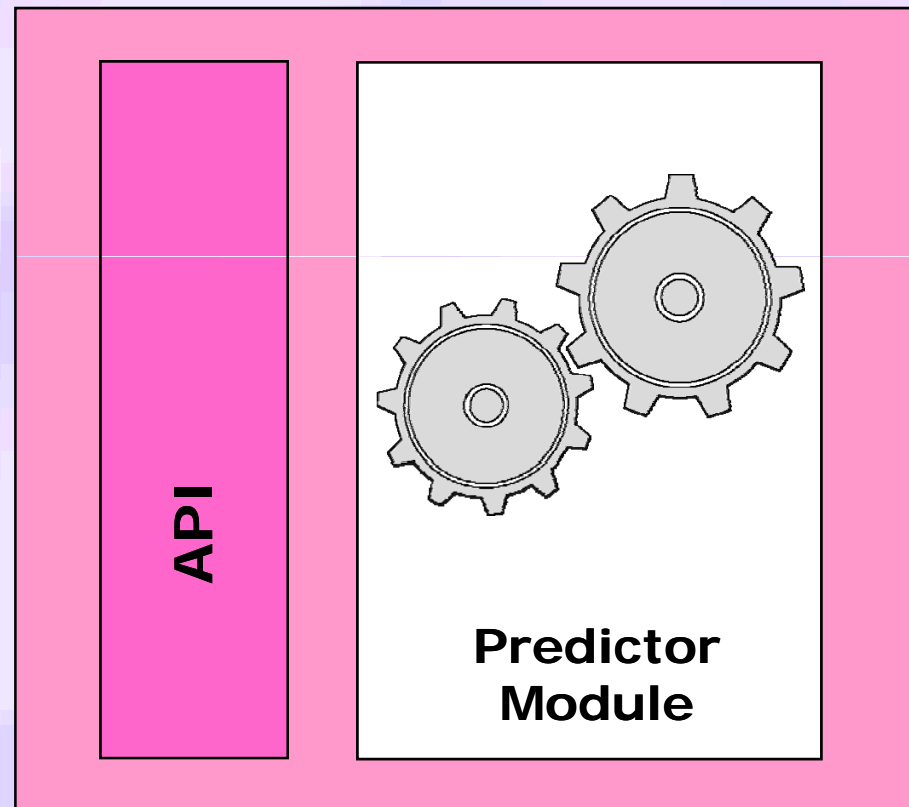
Ideal to know the execution time of each job beforehand



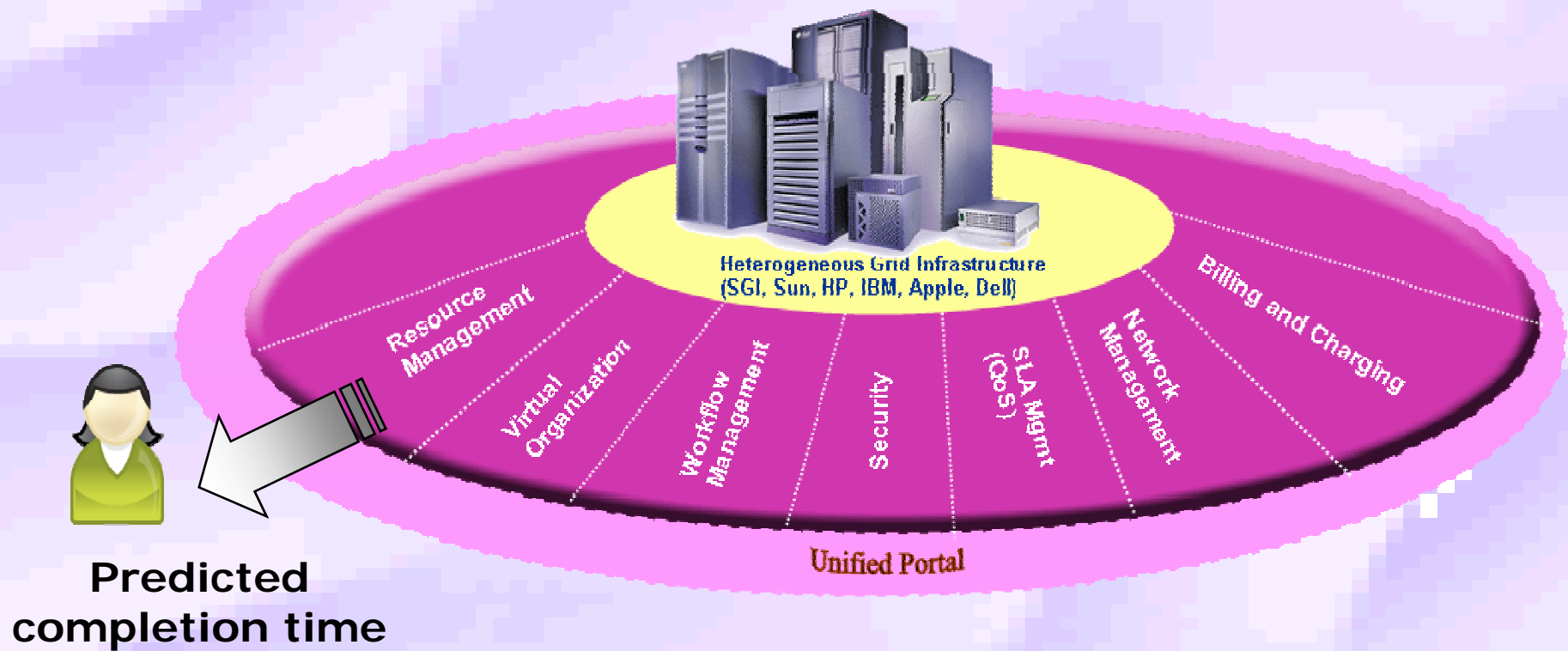
Prediction Module



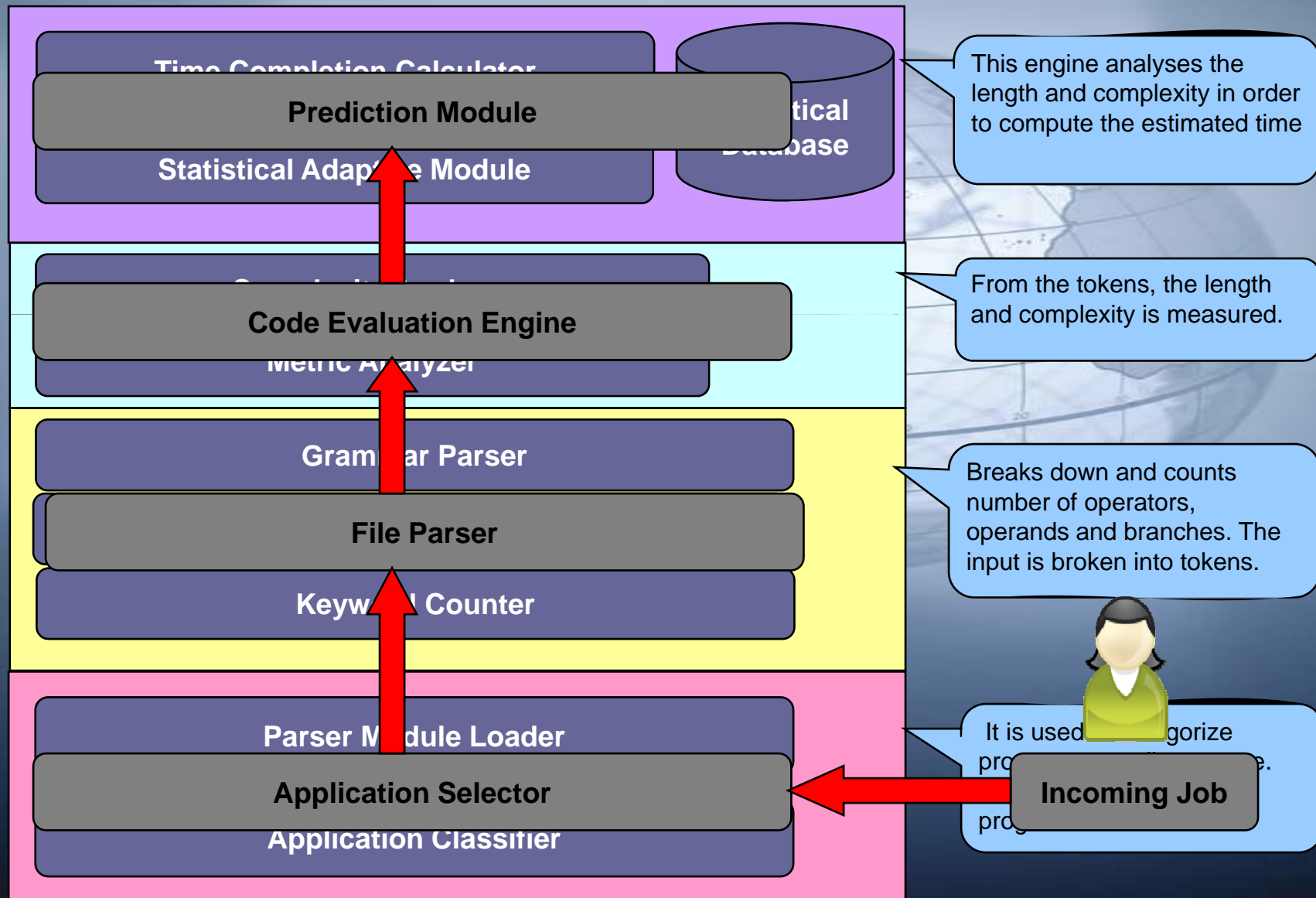
Prediction Module



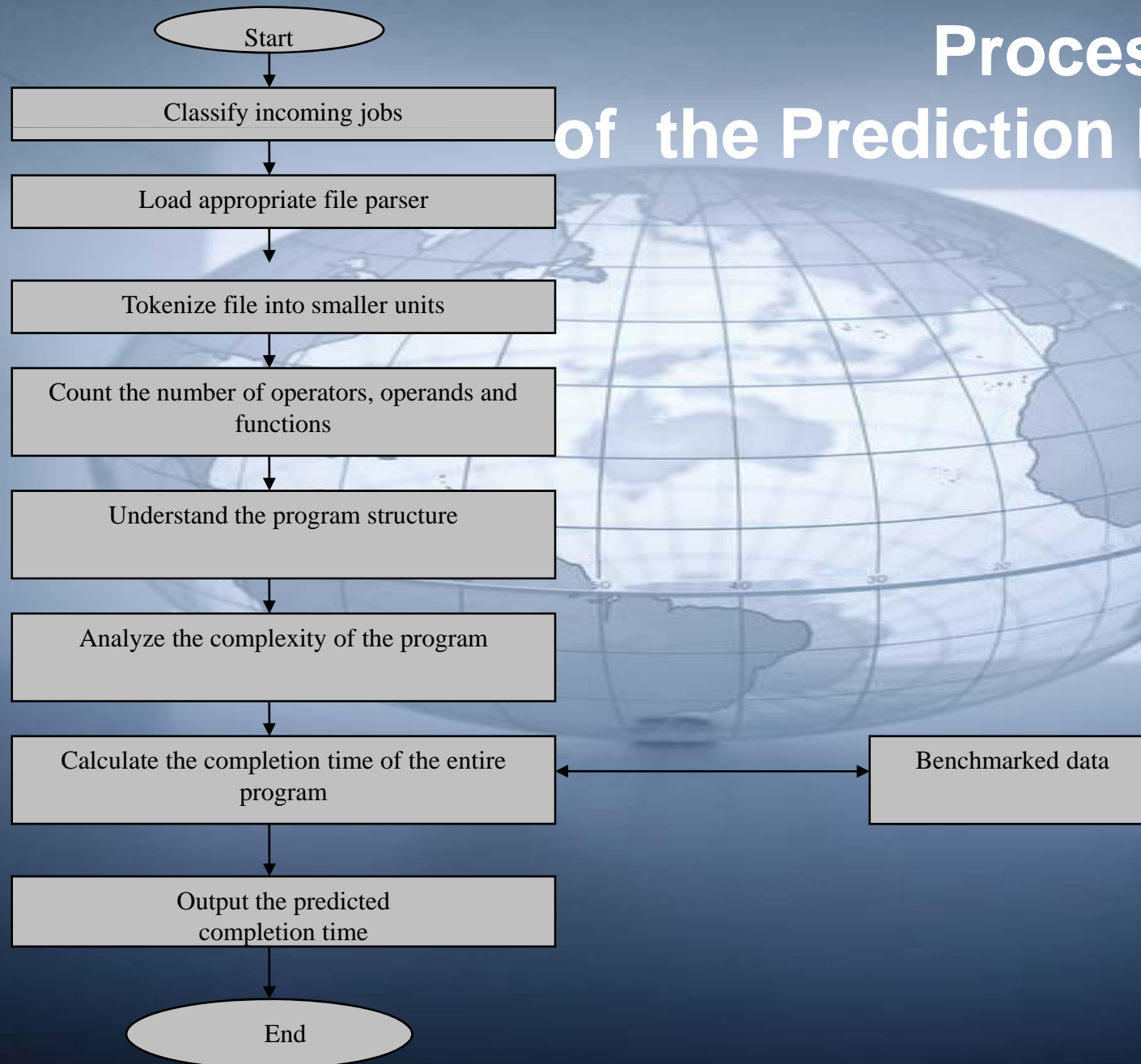
Prediction Module



System Architecture



Process Flow of the Prediction Module



System Architecture

- The proposed architecture is a blend of static analysis, analytical benchmarking and compiler-based approach. Not based on historical data.
- Factors affecting the choice of techniques used in the proposed system:
 - R! programs do not have an identifiable pattern which can be studied to make predictions regarding future incoming job.
 - R! software belongs to the open source community, allowing its code to be freely available and is not bounded by confidentiality agreement .
 - The R application is a system for statistical computation and graphics, involving complex algorithms and computations.

Handling Complexity

Our approach to handling complexity in programs is adapted from three general rules:

- 1. lines of codes (LOC),**
- 2. distinct operators and operands as discussed by Halstead and**
- 3. nesting depth as discussed by McCabe.**

As most non-trivial programs predominantly consist of nested conditional statements or iterations, we focus on handling the complexity of nested loops to improve the prediction accuracy of the module.

Sample Nested Loops

```
for(k in 1:100) { #1
  cat ("\n Loop k: ", k);
  for(i in 1:100) { #2
    cat ("\n Loop i: ", i);
    for(j in 1:100) { #3
      cat ("\n Loop j: ", j);
    } #3
    for(j in 1:100) { #4
      cat ("\n Loop j: ", j);
    } #4
    cat ("\n Loop i: ", i);
  } #2
  cat ("\n Loop k: ", k);
} #1
```

- As the level of nesting grow, the computational step increases as well.
- The complexity increases as the nesting depth increases.

Handling Complexity

1. Break down the nested loops into separate blocks identified by the start line and end line.
2. Compute the execution time, starting with the innermost loop, followed by its parent loop. Repeat until the outermost loop is reached.
3. One iteration of the parent loop includes the complete number of iterations of the child loop.
4. Represented by the following simplified equation:

$$N = \sum_{m=1}^{(l-1)} \left[\prod_{i=1}^m n_i \right] \Delta_m + \prod_{j=1}^l n_j (\Delta_l + P(t_1))$$

Where,

N = total completion time

l = the time taken between the start of one loop and the successive loop

$P(t_1)$ = the amount of time needed to execute the innermost loop.

Implementation Environment

- **The execution time of a program can often vary significantly from one execution to the next on the same system.** This is because computers do not simply execute one program at a time. (dedicated environment)
- **The execution time of one program varies more significantly when it is run across different platform.** This is due to the different specifications of each machine that leads to wide variation in execution time.
- Therefore, all tasks beginning with benchmarking operands and functions to getting the actual executing time are all done using the same machine.
- Similar to other historical-data based prediction, the proposed prediction module is **machine-dependent**. However, the proposed architecture allows cross-platform prediction to be made with **relative ease** as only the **benchmarked scripts** need to be run on the different machines.

Implementation Environment

- **CPU: AMD Athlon 848 (2x2 Core)**
- **Operating System: CentOS 4.4**
- **CPU Speed: 3200 GHz**
- **RAM 2048 MB**
- **Development of Prototype: Java, Eclipse version 3.2.2 as the IDE.**
- **Database: MySQL Server version 5.**
- **The proposed prediction module run on a unix-based environment and is wrapped around an API implementation.**
- **The API was developed using Java Server Page.**

Testing and Evaluation

Two main criteria were used when testing the prediction module:

2. Accuracy of Prediction Test

- Expected to provide an accuracy prediction of 80%, under a Normal Distribution.
- In the Normal Distribution, 68% of the sampling lies within the first standard deviation.

Testing and Evaluation

The mathematical formula to compute accuracy:

Example:

$T_{estimated}$: 100 days

T_{actual} : 110 days

Accuracy error = $(110 - 100) / 110 * 100 = 9.09 \%$

Accuracy = $100 - 9.09 = 90.91\%$

Hence, we can say that this is a successful estimation as the accuracy of prediction is 90.91%, which is more than 80% (threshold goal).

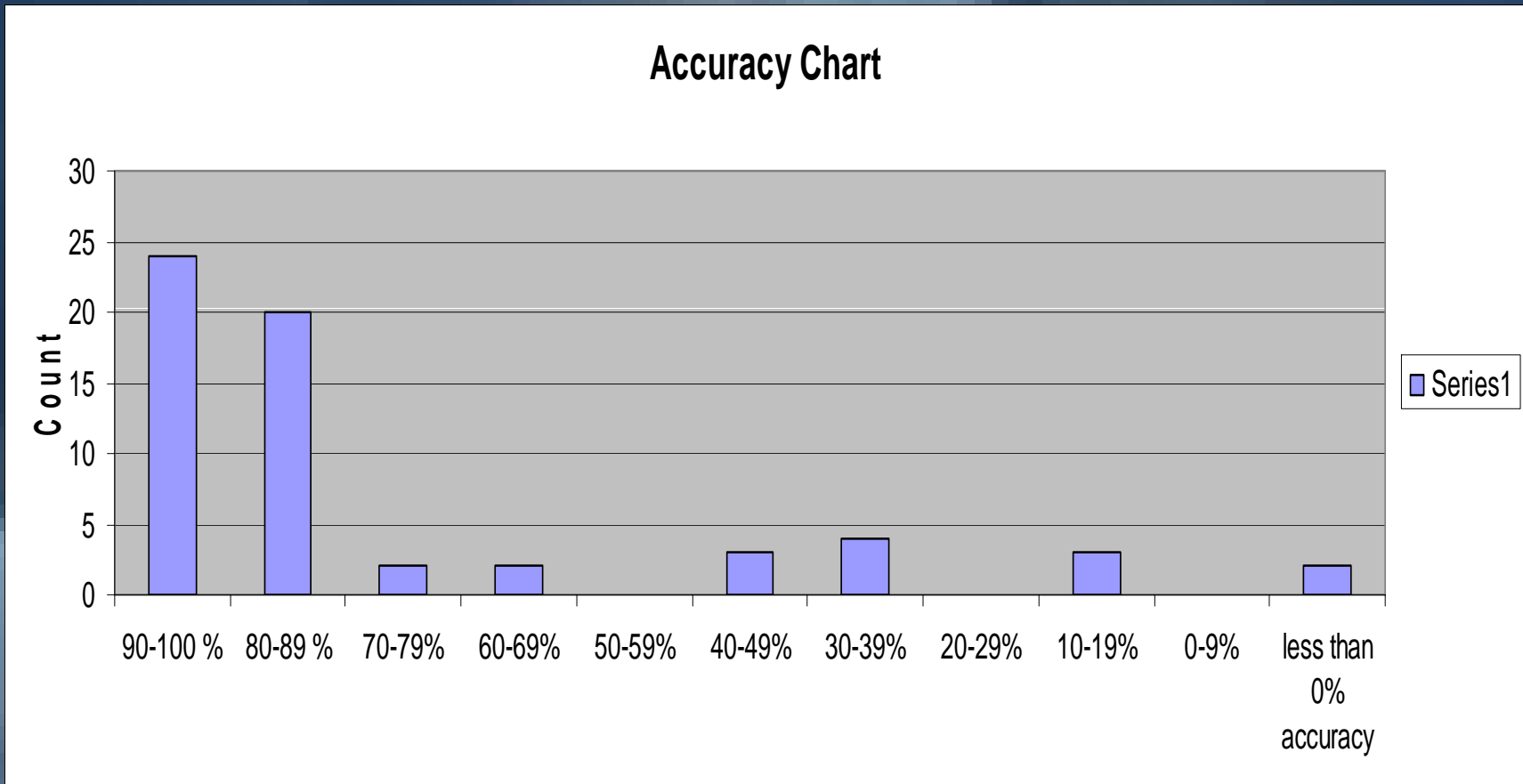
$T_{estimated}$: Time that was estimated by the prediction module

Under Normal Distribution with one sigma of error,
for a total sample of 100 test cases,
the prediction module should be able to predict > **68 jobs** with >**80%**.

Testing and Evaluation

- **At this phase, a total of 60 R! scripts were taken at random as the test cases.**
- **The estimated time to run each of these 60 test cases was predicted using the wrapper developed.**
- **The predicted time was then compared with the actual execution time for each job.**
- **Only predicted time that falls within the range of 80%-120% of the actual execution time is considered successful.**

Summarized Result of Testing Phase



Graphical Representation of Result

Conclusion

- The test results meets the threshold goal of the project.
- Successful as a proof of concept, though it is limited to predicting the execution time of programs written using R! only.
- A solution to this problem will lead to improvement in advance scheduling for resource allocation as well as hasten the transition of the research-driven grids to commercial grids.
- There are still limitations and constraints that can be further refined in the later stage of the project.

Limitations

- **Machine-dependent, platform specific. Benchmarked scripts will have to be run on each machine to obtain the benchmarked execution time.**
- **Does not cover programs written using the object-oriented programming language such as Java and data-feed oriented programs such as Blast.**
- **Dedicated environment. Assumed that jobs will not be pre-empted or interrupted by other jobs.**

However the architecture, methods and process applied are flexible enough to be adapted and applied to other jobs written in various imperative paradigm.

Future Work - In EGEE Context

- The results will be used as a stepping-stone to investigate the building of similar modules for applications being used by EGEE users.
- To offer users a facility to predict execution time of jobs and to express this as a requirement when submitting their jobs to the Grid.
- To assist the Workload Manager to efficiently distribute and manage Grid resources for incoming jobs.

Future Work

- The prediction module should be tested and evaluated for jobs other than R in the near future.
- To improve the efficiency of the prediction, this approach could be refined to handle while loops and dynamic number of iterations.
- Aside from that, the benchmarking approach could be refined or extended further to increase the accuracy of prediction.



**‘The computing field is always in need of new clichés’
-Alan Perlis-**



The background of the slide is a photograph. In the foreground, there is a large, vibrant red star-shaped plant, possibly a species of Dianella, with several long, narrow leaves radiating from a central point. The plant is in sharp focus. In the background, there is a stone building with a visible window and architectural details, slightly out of focus. The overall scene is outdoors with natural lighting.

Thank You

By: Lim Mei Kuan

lim.mkuan@mimos.my

+60 12 633 7702