MONASH University
Information Technology

FIT3142 Distributed Computing

Topic 11: The Design of Distributed Applications

Dr Carlo Kopp
Faculty of Information Technology
Monash University

Why Follow a Systematic Design Approach?

- Many early and existing distributed applications were adaptations and ports of existing applications written for individual machines or clusters;
- As a result some adaptations may be cumbersome or not deliver their potential in performance or functionality;
- *For a application to perform well in a distributed environment its should be architected from the outset to exploit the strengths of a distributed environment and avoid its weaknesses;*
- Therefore a systematic design approach is essential for good results.

Basic Design Considerations [Grid Case Study]

- **Ferreira (IBM):**
- ***"Grid computing is an evolutionary step in distributed computing."***
- ***"Grid computing allows a pool of heterogeneous resources both within and outside of an organization to be virtualized and form a large, virtual computer."***
- ***"This virtual computer can be used by a collection of users and/or organizations in collaboration to solve their problems."***
- ***Key point: "Virtualising" computing resources so they appear as a single "virtual computer".***

Grid Application Design Scenarios (Ferreira 6.)

- **Scenario A: "Clean Sheet of Paper" – permits optimised design at every stage and best possible design choices, but can be very expensive.**
- **Scenario B: "Grid Enabling Existing Application" – may present difficulties dependent on application behaviour; design must accept limitations of existing code; also known as "wrappering".**
- **In Scenario B it is usually industry practice to perform a "qualification" of the application to determine whether it is a good candidate for use in a grid environment. If the application fails the qualification, Scenario A may be the only choice.**

Application Qualification - Fails (Cite Ferreira 6.2)

1. High inter-process communication between jobs without high speed switch connection (for example, MPI); in general, multi-threaded applications need to be checked for their need of inter-process communication.
2. Strict job scheduling requirements depending on data provisioning by uncontrolled data producers.
3. Unresolved obstacles to establish sufficient bandwidth on the network.

Application Qualification - Fails (Cite Ferreira 6.2)

4. Strongly limiting system environment dependencies for the jobs.
5. Requirements for safe business transactions (commit and roll-back) via a grid. At the moment, there are no standards for transactions on grids.
6. High interdependencies between the jobs, which expose complex job flow management to the grid server and cause high rates of inter-process communication.
7. Unsupported network protocols used by jobs may be prohibited to perform their tasks due to firewall rules.

Why Employ Standards in Grid App Design?

- Avoid “reinventing the wheel problem”, always better to make use of existing middleware, interfaces and exploit existing intellectual investment and capital (OGSA, SOA, WSDL, etc).
- Application robustness and reliability – existing standardised middleware and utilities will be be more mature with a large pool of active users; therefore less potential for bugs and interface problems.
- Long term maintainability – standards based code is more accessible to programmers without specific prior experience coding that application.

Basic Steps in Systematic Design for Grids

1. Analyse the application and its intended functionality, and identify what performance needs/expectations exist, or are necessary.
2. Identify key algorithms / datasets and analyse their performance impact (e.g. partitioning problem).
3. Define the application and produce a design specification.
4. Implement the application and code it.
5. Test the application (alpha, beta test) and assess performance.
6. Deploy the application and maintain it.

Why "Ad Hoc" Design is Dangerous

- Grid and web services environments are unforgiving, in the sense that bugs or code design problems are multiplied by the number of processors in the grid, and because networks introduce often large performance penalties.
- If bad choices in application architecture are made early in a design, for instance to make development easier or cheaper, the cost in effort and time to correct these may be prohibitive later in the development cycle.
- The best strategy is always to identify potential problems in robustness/performance before coding .

Basic Design Questions

- It is often helpful to start the development cycle with two basic questions:

1. Why are we building this application to run on a grid? Is there a fundamental reason why it will perform better on a grid or not?

- Ans: Performance? Robustness? User needs?

2. What are the greatest sensitivities this type of application might have to network performance?

- Ans: Delay and network latency? Volume of traffic? Network dropouts?
- *Different applications have different answers!*

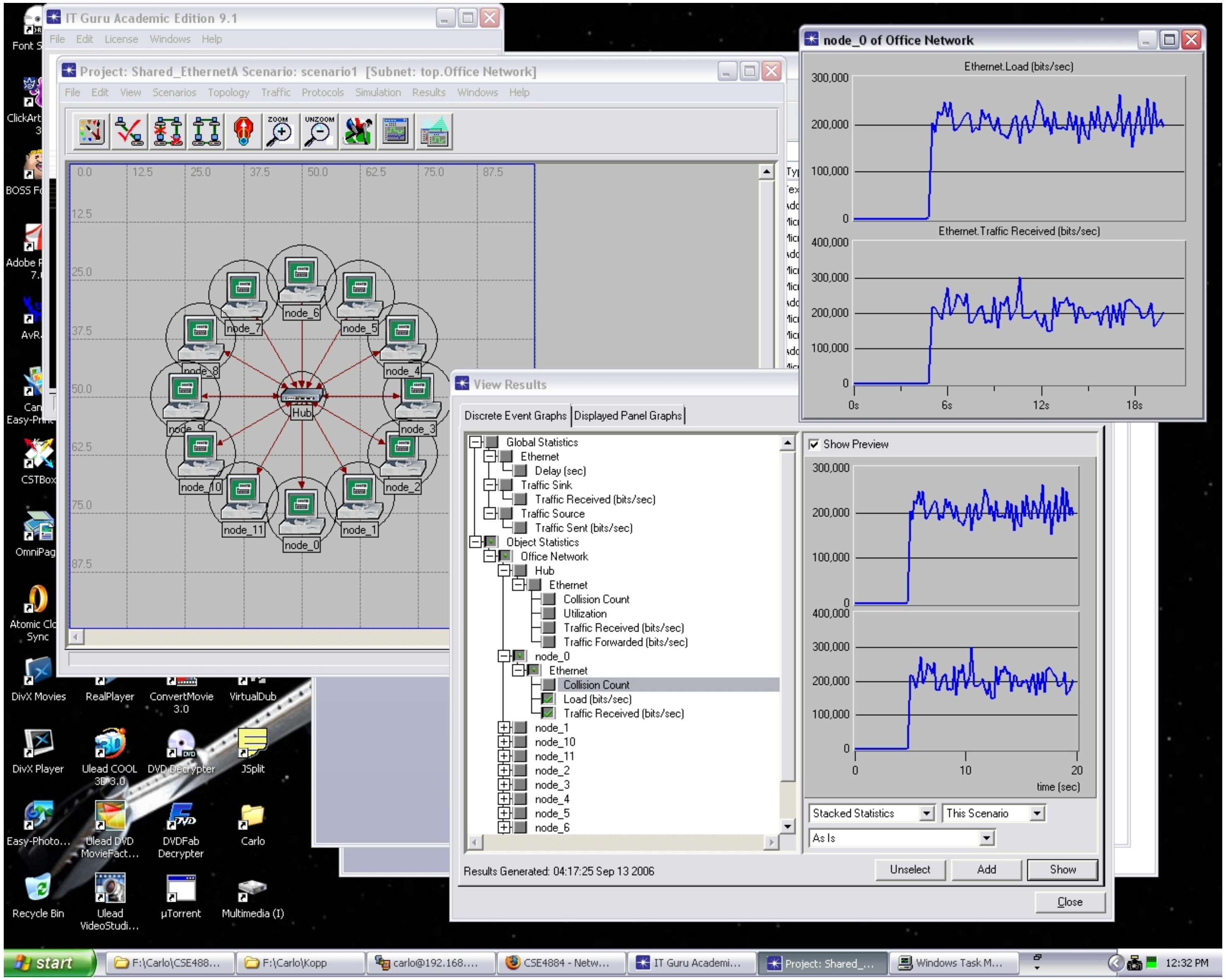
Architecting the Application

- Start with a topology map or sketch for a likely grid to be used and identify what application functionalities must be executed where.
- Identify what messages or traffic must be exchanged between processors running the application.
- Analyse / model / estimate the performance impact of messaging for this architecture?
- Will the application perform adequately against the spec?
- Is more modelling or benchmarking needed?

Modelling and Benchmarking

- If the traffic load expected to be produced by the application is high, then there are numerous alternatives available for assessing its impact.
- While analytical modelling is possible, a more practical approach is the use of a network simulation software tool.
- Examples of such tools are *OpNet* and *ns2*, which are widely used by network designers to simulate traffic loads.
- A good simulator will allow you to construct a topology model for the network, and measure network performance for a range of traffic loads.

Example – OpNet Simulation Model

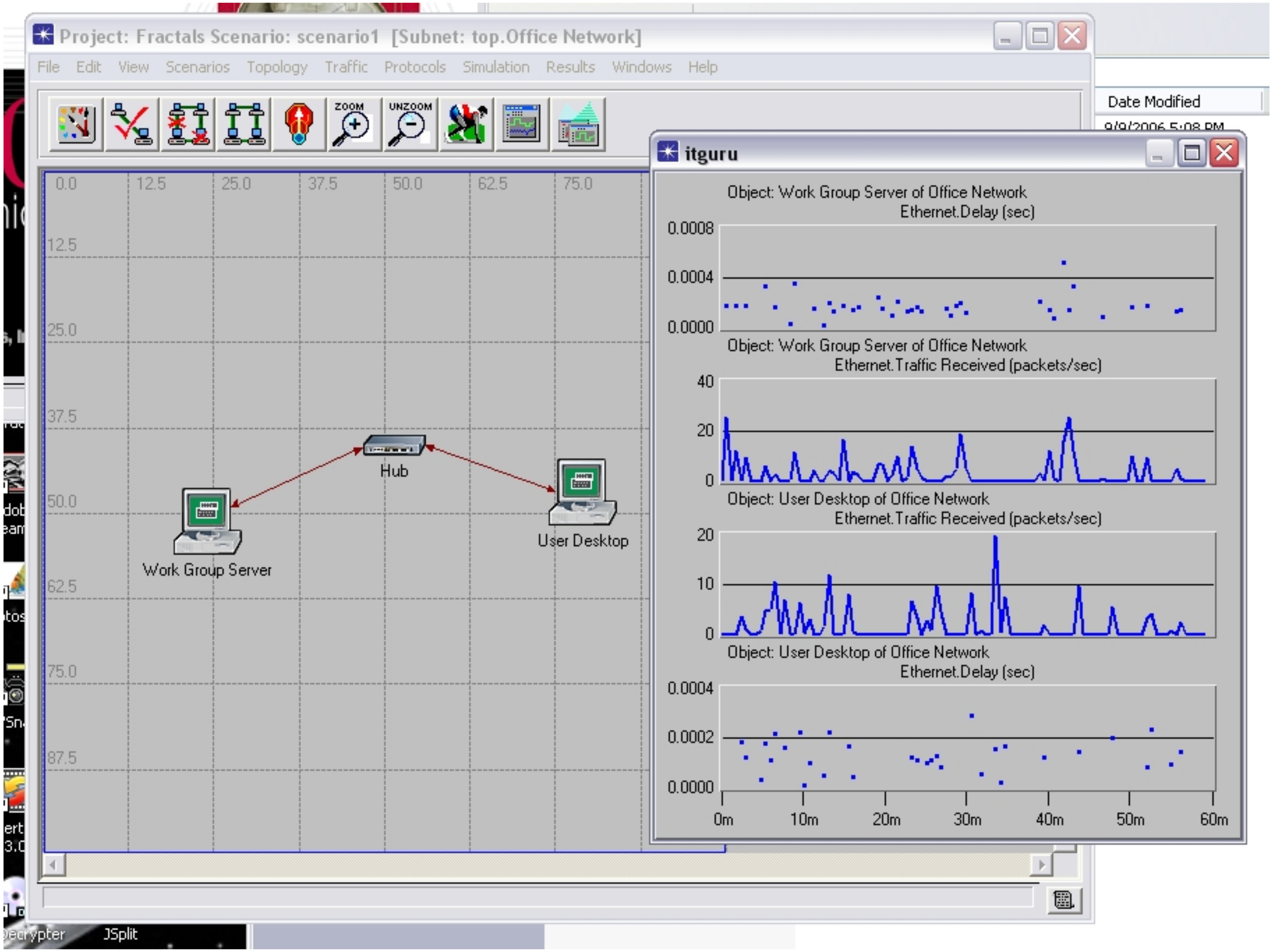


- Global Statistics
  - Ethernet
    - Delay (sec)
  - Traffic Sink
    - Traffic Received (bits/sec)
  - Traffic Source
    - Traffic Sent (bits/sec)
- Object Statistics
  - Office Network
    - Hub
      - Ethernet
        - Collision Count
        - Utilization
        - Traffic Received (bits/sec)
        - Traffic Forwarded (bits/sec)
    - node_0
      - Ethernet
        - Collision Count
        - ✓ Load (bits/sec)
        - ✓ Traffic Received (bits/sec)
    - node_1
    - node_10
    - node_11
    - node_2
    - node_3
    - node_4
    - node_5
    - node_6

Stacked Statistics | This Scenario | As Is

Results Generated: 04:17:25 Sep 13 2006

Example – OpNet Simulation Model



Project: Fractals Scenario: scenario1 [Subnet: top.Office Network]
File Edit View Scenarios Topology Traffic Protocols Simulation Results Windows Help
ZOOM
UNZOOM
Hub
Work Group Server
User Desktop
Date Modified
itguru
Object: Work Group Server of Office Network
Ethernet.Delay (sec)
Object: Work Group Server of Office Network
Ethernet.Traffic Received (packets/sec)
Object: User Desktop of Office Network
Ethernet.Traffic Received (packets/sec)
Object: User Desktop of Office Network
Ethernet.Delay (sec)
0m
10m
20m
30m
40m
50m
60m

Modelling and Benchmarking

- With simulator of suitable quality and good model for the network traffic it is feasible to produce good estimates of transmission delays incurred by grid messaging traffic.
- Important considerations when producing a model include:

1. How much third party or “background” network traffic will be sharing network capacity with grid messaging traffic?
2. What is the volume and type of grid messaging traffic?
3. Is the simulated traffic a good representation?

Modelling and Benchmarking

- Once the messaging delays have been modelled and understood, they can be used to assess the performance impact of the network on the proposed grid application.
- If the performance penalty is low, then the design can proceed to the definition of the software specification, coding and testing.
- If the performance penalty is high, then more analysis is needed.
- Performance problems can be inherent in the algorithm used (poor scalability) or a product of the network intended to be used, or both.

Dealing with Network Performance Impact

- If the problem is a result of poor algorithm performance, then it may be impossible to fix, or difficult to fix.
- Is a different algorithm, which scales better, available for this application?
- If yes, it will be better to rearchitect the application to use an algorithm which scales better.
- The most difficult situation is where the algorithm does not scale well, and there are no good alternative algorithms.
- In this situation the network is the only choice for performance improvements in the grid.

Dealing with Network Performance Impact

- What options are available then?

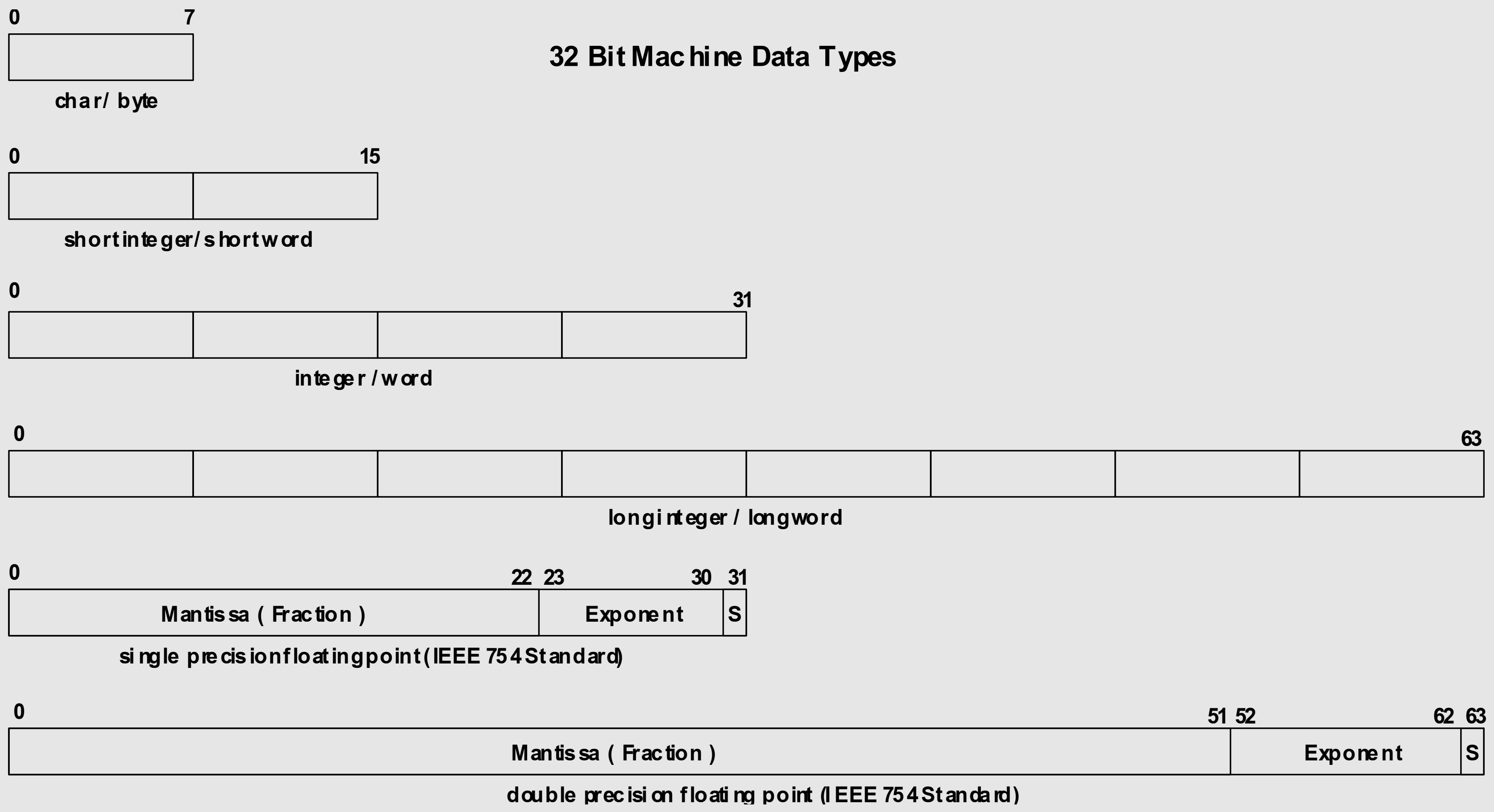
1. Use network QoS facilities if available, this may or may not solve the problem.
2. Upgrade the network so it can carry more traffic with less delay, which is expensive.
3. Explore the messaging strategy used by the algorithm to determine whether changes to the message format can improve performance.

- Example: if the data can be represented in 8 or 16 bits rather than a native 32, 64 or 128-bit format, then an N x M matrix of data can be made much smaller for transmission.

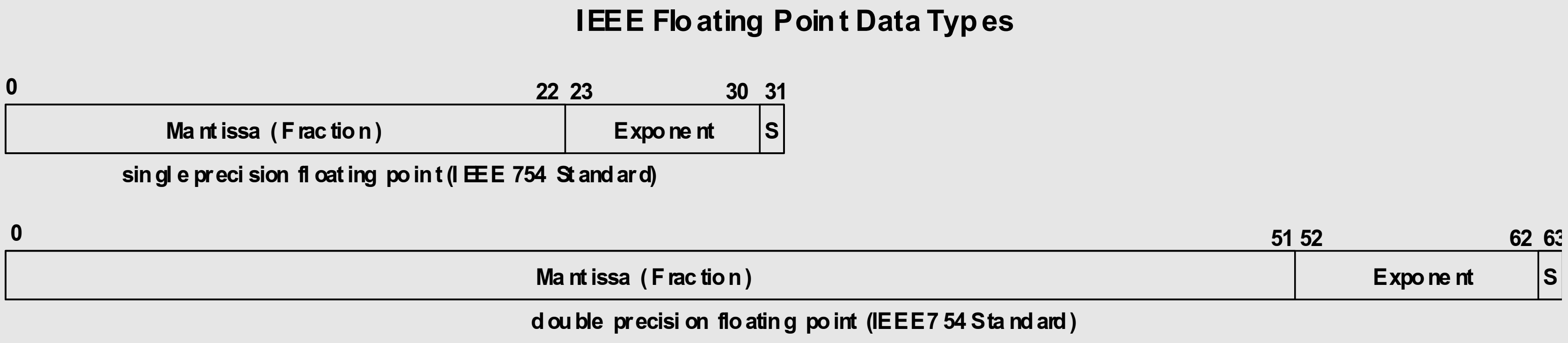
Compacting Message Data

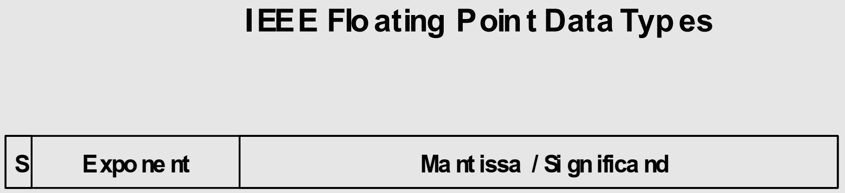
- Consider a situation where a 2D or 3D mesh computation is to be performed on a grid.
- Each node in the computation must exchange messages with 4 or 6 neighbouring nodes.
- Each message comprises a 2D array i.e. N^2 values.
- Each value is bounded in size from above and below.
- The default computation uses a double precision floating point format (64 bits).
- Is double precision necessary for messaging?
- If yes, we cannot compact the message format.

Data Representation 32-bit Machine



Floating Point Data Representation


IEEE Floating Point Data Types
0
22 23
30 31
Mantissa (Fraction)
Exponent
S
single precision floating point (IEEE 754 Standard)
0
51 52
62 63
Mantissa (Fraction)
Exponent
S
double precision floating point (IEEE 754 Standard)



Value = Sign Bit * (Mantissa * 2 ^ Exponent)

Data Representation – C/C++

- *"char"* = 1 byte = 8 bits = 8 D-latches = 1 ASCII character
- *"int"* = 1, 2, 4, 8 bytes = 8, 16, 32, 64 bits depending on machine.
- *"short"* = usually one half of "int" e.g. 1, 2, 4, 8 bytes ...
- *"long"* = usually two "int" i.e. 2, 4, 8, 16 bytes ...
- *"float"* = single precision floating point.
- *"double"* = double precision floating point.

Data Representation – C/C++

```
void main{}
{
char value1;    /* ASCII character value */
short value2;   /* short integer value */
int value3;     /* integer value */
long value4;    /* long integer value */
float value5;   /* floating point value */
double value6; /* double precision floating point value */
fprintf( stdout, "hello world\n");
} /* end main */
```

Compacting Message Data

- **If single precision is acceptable for messaging, then how much do we save by using 32-bit single precision vs 64-bit double precision, given N^2 data elements in a message?**
- **How many bytes are required if we convert from double to single precision, per message?**
- **Given message size is 8 bytes for each "double" and 4 bytes for each "float", then 8/4 = 2; 2^2 = 4.**
- ***By using a single precision rather than double precision value in the message, we have reduced the size of a N^2 data element message four fold.***
- ***Are there other techniques possible?***

Compacting Message Data

- In some computations the messages which are exchanged may involve sparse matrices, or repeated patterns of numbers.
- Such behaviours, if repeated over and over again, can be exploited to reduce message size.
- In such situations, it is feasible to set up the messaging so that what is transmitted comprises the parts of the message which change message by message.
- This idea is used for instance in VJ TCP header compression, for low speed PPP links.

Compressing Message Data

- Another technique for reducing message sizes is that of compressing messages before transmission, and uncompressing them after receipt.
- Compression and decompression is usually more expensive computationally, compared to compacting data.
- However, compute power is relatively cheap, and in a grid application where communications delay is very expensive, compression may work well.
- There are many good compression algorithms and mature well tested code available.

Key Caveats for Grid Application Design

- Algorithm scalability – is the algorithm sensitive to network delays, what data dependencies exist?
- If the algorithm is sensitive to network delays, multiple strategies available:

1. Use middleware QoS functions
2. Use faster network
3. Minimise message frequency if possible
4. Compact message data
5. Compress message data

- *These problems must be solved before code is written, else the results might be catastropic.*

Conclusions

- Application design for the distributed environment is much more complex than conventional application design for single processor systems, or conventional multiprocessor systems.
- Whether a new application is being designed, or an existing application is to be converted to run in a distributed fashion, it is necessary to analyse the application's behaviour to establish the impact of network performance.
- Some algorithms/applications are not suitable for distributed operation.
- Many techniques exist to improve performance.

Reading / References

- Ferreira L et al, *Grid Services Programming and Application Enablement*, Red Book, International Business Machines, International Technical Support Organization, May 2004, URL: http://www.redbooks.ibm.com/abstracts/sg246100.html
- Ferreira L et al, *Introduction to Grid Computing with Globus*, Red Book, International Business Machines, International Technical Support Organization, September 2003, URL: http://www.redbooks.ibm.com/abstracts/sg246895.html