Desktop Grid Computing

Ashok Adiga
United Devices, Inc., Austin, Tx
Ashok@ud.com
(512) 294-4198

Outline

- Introduction to Desktop Grid Computing
- UD Metaprocessor Overview
- Desktop Grid Applications
- Programming on the Metaprocessor
- Conclusions

What is the Grid?

"Software and technology infrastructure to support the coordination and sharing of resources in dynamic, distributed virtual organizations."

From "Physiology of the Grid", by Foster, Kesselman, Nick & Tuecke

What does this mean?

- **Resources can be compute cycles, data, applications, storage and network IT resources**
- **Virtual organizations can be intra-enterprise or inter-enterprise**
  - Departmental boundaries not necessarily defined by location
- **Desktop Grid computing has always been about coordinating resources for "virtual" supercomputing**

Typical Grid Computing Resources

- **Supercomputers**
  - High Speed, Reliable, Very Expensive
  - Low overhead communication (typically shared memory)
  - 10s of CPUs
- **Clusters**
  - High Speed, Reliable, Moderately Expensive
  - Low overhead communication (custom connections, Myrinet, SP switch)
  - 100s of CPUs
- **Desktop PCs and Workstations**
  - Low Speed (but improving!), Heterogeneous, Unreliable, Non-dedicated, Inexpensive
  - Generic connections (Ethernet connections)
  - 1000s-10,000s of CPUs

Desktop Grid Challenges

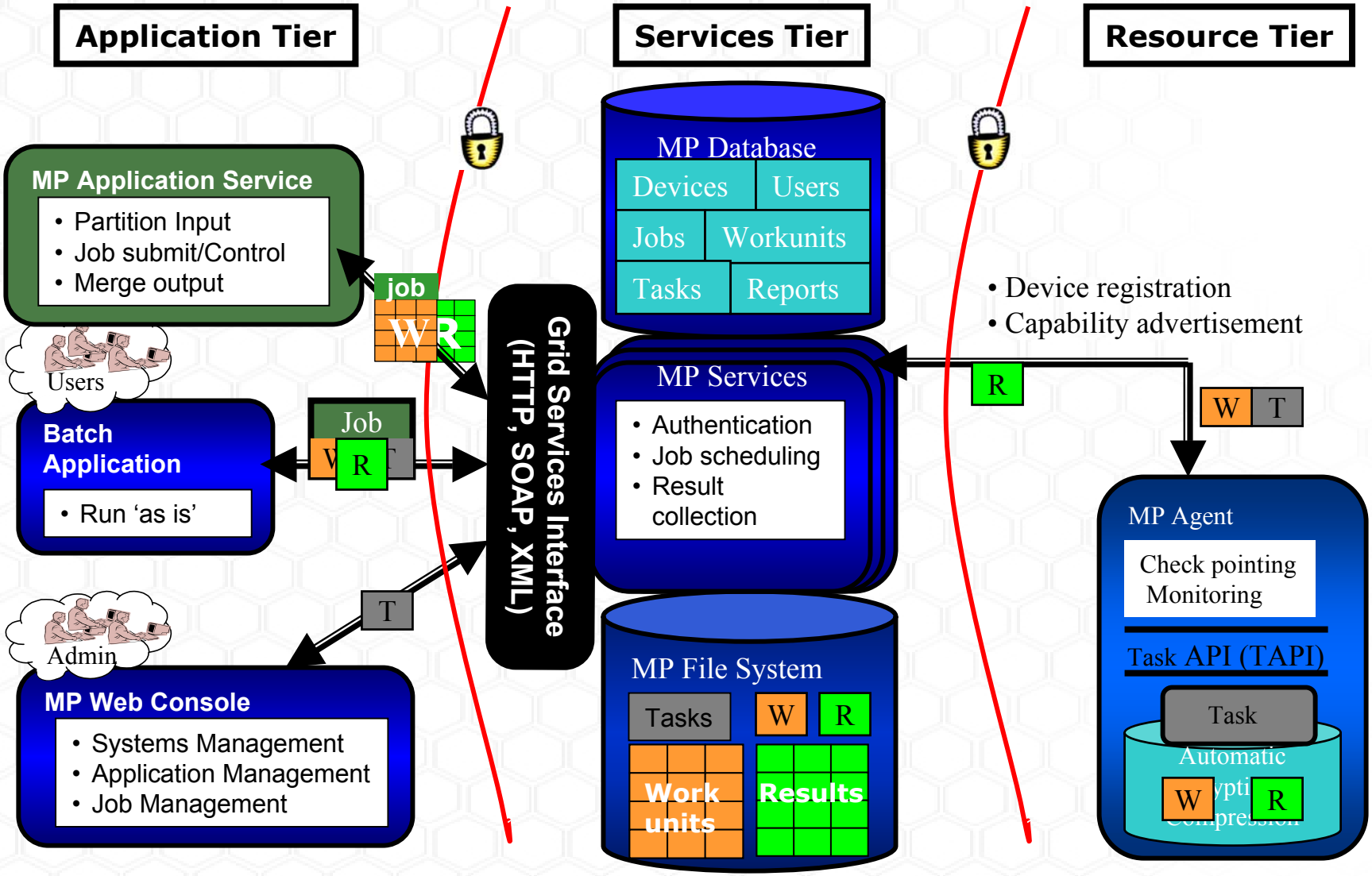
- **Scheduling heterogeneous, non-dedicated resources**
- **Added Security requirement**
  - Desktop machines typically not in secure environment
- **Unobtrusiveness**
  - Harness underutilized computing resources without impacting the primary Desktop user
- **Connectivity characteristics**
  - Not always connected to network
  - Might not have fixed identifier (IP address)
- **Limited Network Bandwidth**
  - Ideal applications have high compute to communication ratio
  - Data management is critical to performance
- **Fault Tolerant**
  - Machines are typically more unreliable than clusters
- **Interoperability**
  - Must adhere to Grid standards

Existing Desktop Grid Solutions

- **Condor**
  - University of Wisconsin
  - 10+ years
  - Initially targeted at scheduling clusters, now supports Desktop PCs
- **Entropia**
  - Startup based in San Diego
  - 2+ years old
  - DC Grid Platform (commercially available)
- **Platform Computing**
  - Startup based in Canada
  - 10 years old
  - LSF, mainly targeted at scheduling clusters
- **United Devices**
  - Startup based in Austin
  - 2.5 years
  - Metaprocessor Platform (commercially available)

UD Metaprocessor Overview

MetaProcessor Architecture



Metaprocessor Architecture

MP Console

MP Application Service (CLI/GUI)

MetaProcessor Grid Services Interface (SOAP/XML-RPC)

MetaProcessor Functional Areas

Device Management	User Management	Application Management	Archiving
Data Management	Workload Management	Accounting Reporting	Resource Management

DBMS
DB2
Oracle

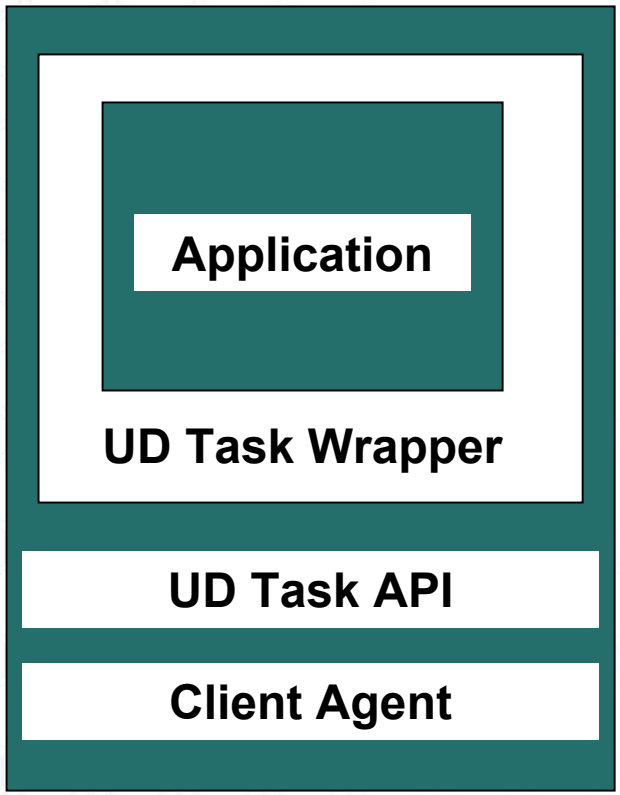
MetaProcessor Infrastructure Services

Authentication Service	File Service	Dispatch Service	Device Poll Service

MP Agents

Metaprocessor Client Architecture

- Application is encapsulated with Task Wrapper to obtain a Task Module
- Client agent is responsible for:
  - Managing execution of task module
  - Encryption of input/output files for application
  - Communications with the UD server
  - Monitoring and reporting client characteristics
  - Enforcing device preferences
  - Miscellaneous housekeeping
- Task Module communicates with UD agent using the UD Task API, a published interface (optional)
  - Checkpointing
  - Task monitoring



Features: IT Administrators

Comprehensive Security

- **Authentication** – validates users & devices
- **Triple-DES encryption** – protects data on the network and disk
- **Checksum & digital signature** – tamper-proofs data from modifications

Device Unobtrusiveness

- **Idle time execution** – prevents disruption of work on the desktop
- **Device usage preferences** – ensures resources are used only when they are needed

Platform Heterogeneity

- **Linux & Windows** – covers most widely used operating systems in the enterprise

Advanced Workload Scheduler

- **Priority, fairness and resource availability** – optimizes allocation of resources to jobs
- **Applications and resource constraints** – ensures that jobs are run on devices capable of running them

Features: IT Administrators

High Scalability

- **Industry standard architecture** – scales to millions of resources with a single installation

Superior Manageability

- **Web-browser console** – enables remote access for both users and administrators
- **Self-updating infrastructure** – ensures upgrades without disrupting desktop users
- **Centralized infrastructure** – simplifies maintenance and control

Performance Optimizations

- **Node data caching** - enables low network bandwidth consumption
- **Data compression** - reduces storage and network usage
- **Data Affinity scheduling** – optimizes data distribution and network usage

Features: Application Developers

- **Industry standards**
  - **Web services, XML & SOAP interfaces** – simplifies application migration and integrates easily into existing IT environments
  - **GGF sponsorship** – enables UD to incorporate important customer features into the OGSA standards
- **Rapid Application Migration**
  - **Programming & command line interfaces** – provides users with a variety of usage options
  - **Development toolkit** – enables developers to quickly understand the technology and migrate applications

- **Flexible Job Specification**
  - **Batch and data-parallel jobs** – enables a large set of applications on the MetaProcessor platform
  - **Redundancy** – ensures job completion in hostile environments
  - **Runtime limits** – prevents 'run away' jobs

- **Application Lifecycle Control**
  - **Versioning** – simplifies application maintenance
  - **Phases** – ensures deployment of well tested applications

Features: Application Users

- Customizable User Interfaces
  - **End-user transparency** – enables users to continue using their favorite interfaces
- **Performance Improvements**
  - **Throughput** – jobs from multiple users complete much faster
  - **Response time** – improves the performance of a single job by nearly 100X
- Application Data Management
  - **Data registration and naming** – enables users to store data in the MetaProcessor, securely share it with others and reuse it across many jobs

Desktop Grid Applications

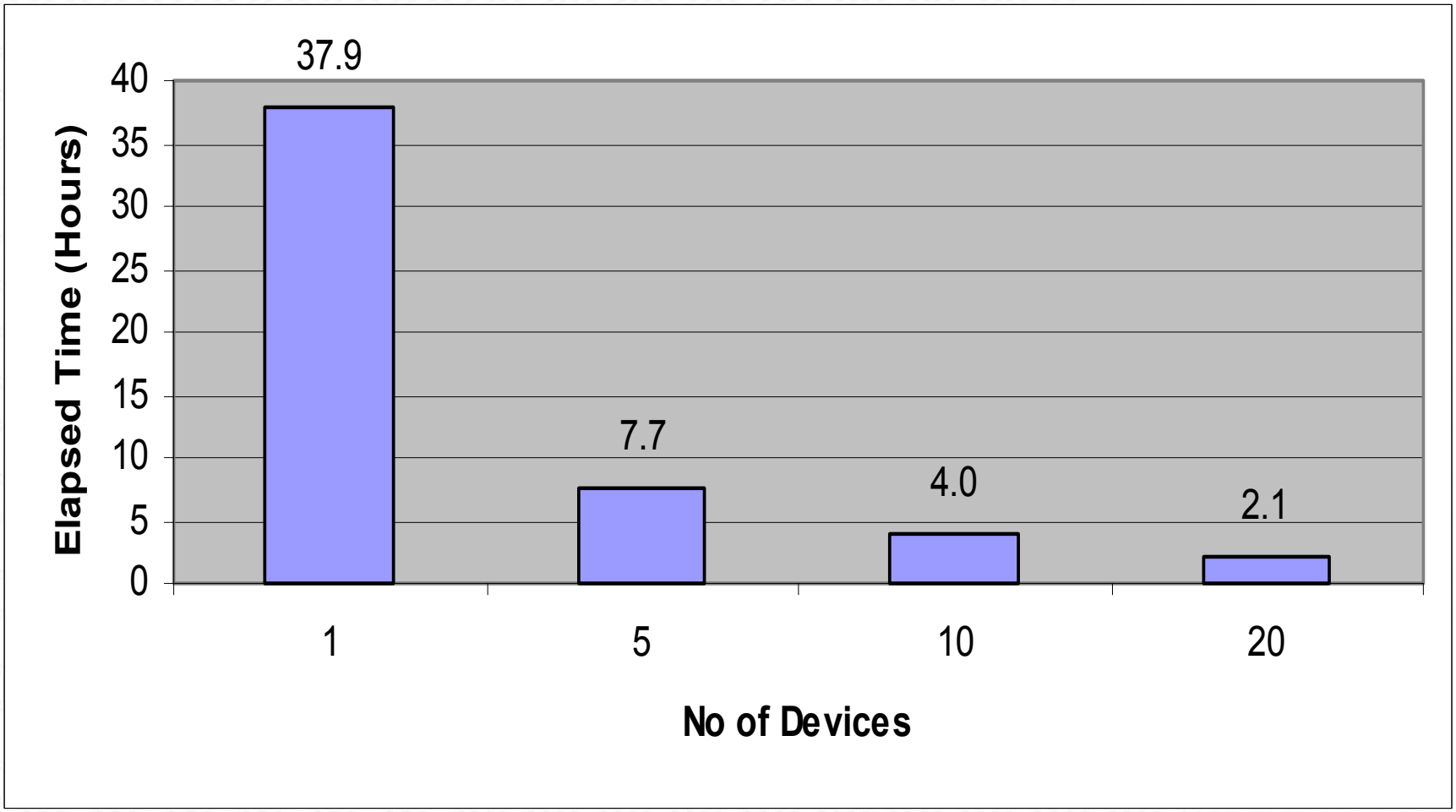
Application Characteristics

- Platform is ideally suited to running large, compute intensive jobs
- Programming model suited to coarse grained parallelism
- Naturally supports Data Parallelism
  - **Monte Carlo methods**
  - **Large Database searches**
  - **Genetic Algorithms**
  - **Exhaustive search techniques**
- Future direction includes P2P extensions to enable fine-grained parallelism
  - **MPI support**
  - **Data caching**

Reservoir Simulation (GeoSciences)

- **Landmark's VIP product benchmarked on MP**
- **Workload consisted of 240 simulations for 5 wells**
  - Sensitivities investigated include:
    - 2 PVT cases,
    - 2 fault connectivity,
    - 2 aquifer cases,
    - 2 relative permeability cases,
    - 5 combinations of 5 wells
    - 3 combinations of vertical permeability multipliers
  - Each simulation packaged as a separate piece of work.
- **Client Devices**
  - 20 Devices, 800 MHz Celeron, 192 MB, 100 mbps network
- **Software**
  - VIP (core and exec), UD MP v3.0

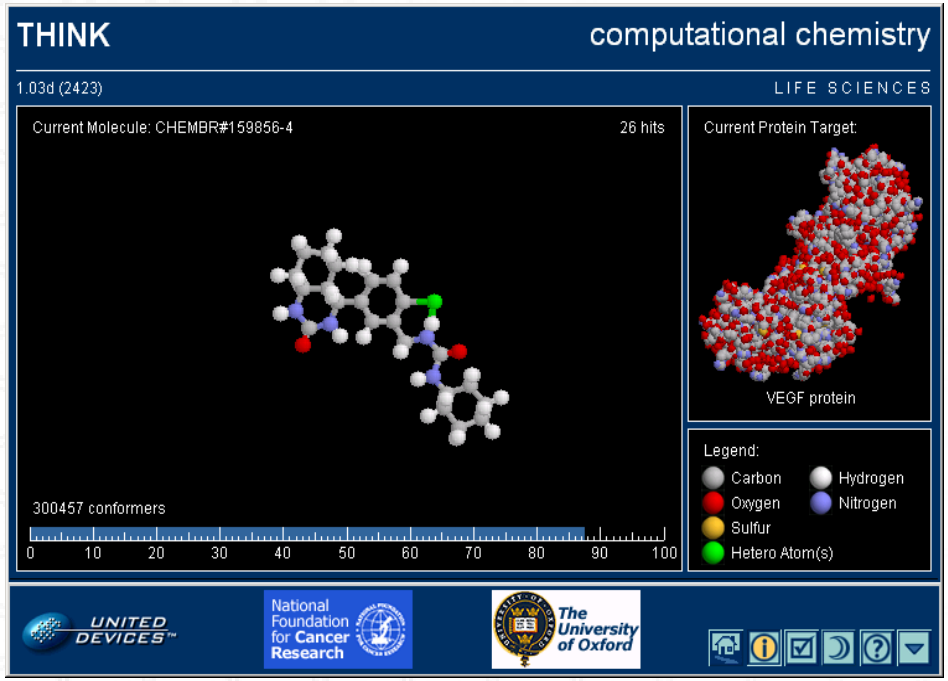
VIP Response Times (measured)



Drug Discovery (LifeSciences)

➢ THINK application

- Internet Project in partnership with Oxford University Model interactions between proteins and potential drug molecules
- Virtual screening of drug molecules to reduce time-consuming, expensive lab testing by 90%
- Drug Database of 3.5 billion candidate molecules.
- Over 350K active computers participating all over the world.



THINK

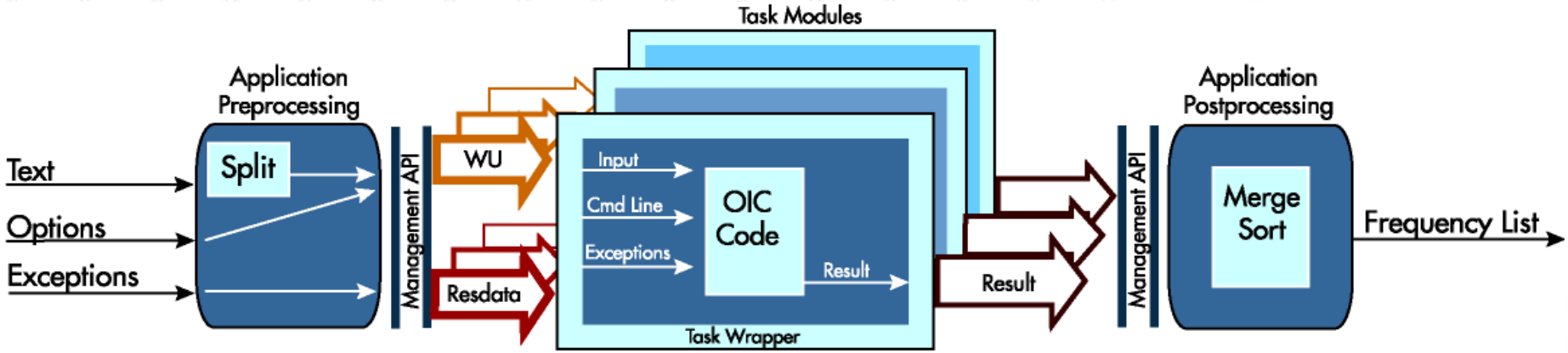
- **Application Characteristics**
  - Typical Input Data File: < 1 KB
  - Typical Output File: < 20 KB
  - Typical Execution Time: 1000-5000 minutes
  - Floating-point intensive
  - Small memory footprint
  - Fully resolved executable is ~3Mb in size.
- **Project now in 2nd Phase**
  - More detailed analysis of “hits” from first phase
  - Ligandfit application from Accelrys

Programming on the MetaProcessor

OIC: Ordered Item Count

- Example task that parses an input text file and builds an ordered index of words in that file
  - **Inputs:**
    - **Text file**
    - **List of words to exclude from the counts**
  - **Outputs:**
    - **Ordered index of words along with frequency of occurrence**
  - **Command line options:**
    - **Inclusion/exclusion; sleep**

OIC Parallelization



- Input text file is split into several parts
- Workdata = Input text file fragment + Options
- Resdata = Exception word list
- OIC code produces individual result lists
- Result lists are merged to create one frequency list

OIC Porting

- **Identify parallelism in application**
  - break up textfile into smaller fragment and process fragments independently
- **Create Taskmodule**
  - wrapper, buildmodule, buildpkg
- **Upload task to MetaProcessor Console**
- **Write Application Scripts**
  - Split input data and merge results
  - Solution with UD MAPI and SOAP
  - two perl utilities to be run from any unix box:
    - ete-submitjob.pl and ete-retrievejob.pl
- **UD TAPI integration (optional)**
  - checkpointing

Creating taskmodules

- **Create the Taskmodule with the ‘`buildmodule`’ utility**
  - buildmodule will bundle your executable with a loader called ‘wrapper.exe’ and a config file ‘mdf.xml’
- **Use ‘`buildpkg`’ to construct workunits/resdatas**
  - buildpkg will bundle your inputfiles into a tar package, with optional compression, and a config file ‘pmf.xml’
- **Try out in the TestAgent**
- **That’s all!**

Task Wrapper

- **Provides support for running executables on MP client:**
  - Command line arguments
  - Environment variables
  - Variable substitution in command line arguments and environment variables
  - Redirection of stdin, stdout, stderr
  - Handling of multiple files in input or output packages
  - Compression of input and output files

Application Scripts

- **Utilize UD MAPI to construct automated solution, called "Application Service"**
  - Management API uses XML-RPC or SOAP standard
  - XML-RPC and/or SOAP client libraries available in C/C++, C#, Java, Perl, PHP and many more…
- **Typical preprocessing:**
  - create new Job in this Task
  - split data, package data with buildpkg
  - submit resdata and workunits to Job through MAPI
- **Typical postprocessing:**
  - get all results back through MAPI
  - do merging of data

MAPI structures and calls

- **Various data entities in the system have XML-RPC/SOAP equivalents**

```
struct Workdata
   int id                  // The unique id of this Workdata record.
   int workdatasetid       // The id of the Workdataset to which this Workdata record belongs. It
                           must be specified non-zero, and must point to an existing Workdataset
                           record. (required)
   int taskid              // The id of the Task to which this Workdata record applies. Needs to
                           correspond with the taskid of the Workdataset this Workdata belongs
                           to. (required)
   string filename         // The name of the Workdata file. (max length 254) (required)
   boolean data_exists // This is true if the 'data' member of this structure contains file data.
                           (required)
   base64 data             // The actual Workdata file data. There are no restrictions on the
                           content of the data.
   int index               // Index field for this Workdata record. (required)
end struct
```

MetaProcessor Management API

- **Various operations have XML-RPC/SOAP 'methods'**
- **Inserting data in the system include:**
  - **addTask** (auth, struct Task)
  - **addResdataset** (auth, struct Resdataset)
  - **addResdatas** (auth, struct[] Resdatas)
  - **addWorkdataset** (auth, struct Workdataset)
  - **addWorkdatas** (auth, struct[] Workdatas)
  - **addJob** (auth, struct Job)
  - **addWorkunits** (auth, struct[] Workunits)

MetaProcessor Management API

- **Retrieving data include methods like:**
  - struct Job **getJob** (auth, jobid)
  - struct JobStatus **getJobStatus** (auth, jobid)
  - struct[] Workunits **getWorkunitsForJob** (auth, jobid)
  - struct[] Results **getResultsForWorkunit** (auth, wuid)
  - struct Result **getResult** (auth, resultid)
  - **deleteResults** (auth, resultid)

OIC example pre-processing in Perl

```
#!/usr/bin/perl -w
use Frontier::Client;
use LWP::UserAgent;

$auth = $server->call("login", "username", "password");
my $job_id = $server->call("addJob", $auth, $jobdef);

# split the input text files
`/usr/bin/split --lines=$splitsize $textfile $textfile-split-`;
my @fragments = glob("$textfile-split-*");

foreach my $fragment (@fragments) {
    # first wrap up the textfragment into a workdata-package
    my $wdfilename = $fragment.".tar";
    `$buildpkg -f -DOTHEROPTIONS=$customoption $wdfilename $fragment=fragment.txt`;

    # upload the workunit-package
    my $submiturl = "http://server/filesvr?auth=$auth&type=workdata&filename=$wdfilename";
    my $request = HTTP::Request->new('POST', $submiturl);
    open(F, $wufilename); $request->content(<F>); close (F);
    my $response = $ua->request($request);

    # add the workdata through the XML RPC interface
    my $workdatatid = $server->call("addWorkdata", $auth,
            { jobid=>$jobid, state=>1, filename=>$wdfilename );
}
```

login

add job

split work

package data

upload package

add workunit

OIC example post-processing in Perl

```
my $jobstatus = $server->call("getJobStatus", $auth, $job_id);
if ($$jobstatus{done_workunits} < $$jobstatus{total_workunits})
    die "job only at $$jobstatus{completed_percent}\n";

# Retrieve all results for every workunit in this job
my %total_frequencies = ();
foreach my $workunit (@$workunits) {
    my $results = $server->call("getResultsForWorkunit",
                    $auth, $$workunit{id}, $server->boolean(0), 0, 0);
    my $result = @$results[0]; # 1st result only used here

    #retrieve result file from MP Server
    my $tempfile = `mktemp -q -u tmpresXXXXXX`;
    my $resulturl = "http://server/filesvr?auth=$auth&type=result&resultid=$$result{id}";
    my $request = HTTP::Request->new('GET', $resulturl);
    my $response = $ua->request($request, $tempfile);

    # add data to total frequency list
    open (RESULT, $tempfile);
    my @wu_frequencies = <RESULT>;
    foreach (@wu_frequencies) {
        my ($freq, $word) = split;
        $total_frequencies{$word} += $freq;
    }
    close (RESULT);
    unlink $tempfile;
}
```

get status

retrieve results

transfers result

merge frequencies

Porting options

- **Basic (just demonstrated)**
  - fast turnaround
  - no code changes needed
  - file I/O systemcall interception for encryption
  - language independent
- **Advanced: use UD Task API**
  - provides checkpointing of the application
  - Requires application code changes
  - provides graceful shutdown, suspension of application
  - available in C, C++, and Fortran

Conclusions

Resources

- SDK available
  - **Tools, Libraries, Documentation**
  - **Helper libraries in C++ and perl for UD MAPI**
  - **Application Developer's Guide (ADG)**
    - **Detailed documentation of Interfaces**
    - **Comprehensive examples**
  - **Webpage at: http://www.ud.com/products/sdk.htm**
- MP Currently deployed at UT
  - **Bob Gloyd's Engineering Labs**
  - **Being taken over by UT-TACC**

Future Directions

- Align with Grid standards as they are accepted
  - **OGSA/Globus**
- Enable P2P features
  - **Aggregate other resources such as storage**
  - **Support hybrid models (MPI, data caching)**
- Enhance data management features
- Define High Level Abstractions for specifying Distributed Applications

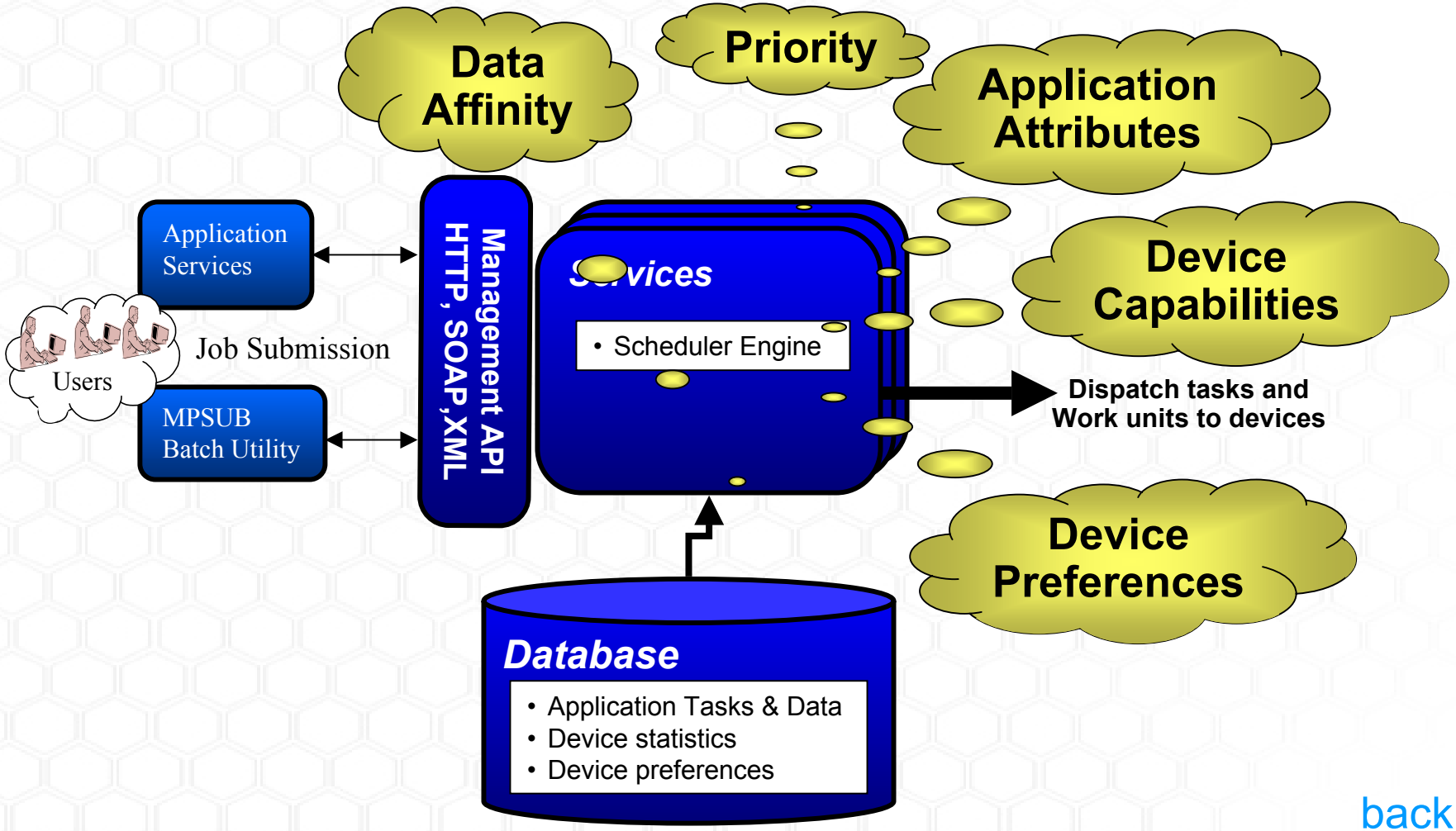
Backup Slides

MetaProcessor Grid Interfaces

- **MetaProcessor Grid Services Interface (MGSI)**
  - XML and SOAP-based programming interface for developing distributed applications
  - First Grid vendor to offer industry standard Web services interface
  - Microsoft .NET-based reference implementation available
  - Supports 22 programming languages
- **Grid Command-line Interface for batch Job Processing**
  - MPSUB command offers rich set of options to submit batch jobs
  - MPRESULT command tracks and controls batch job results
- **Grid Interoperability**
  - MGSI easily is extendable to support emerging OGSA standards
  - Integrates easily with existing

back

Workload Scheduler



back

Job Specification

- **Batch Jobs**
  - MPSUB & MPRESULT utilities to submit, control and obtain results for batch jobs
  - Program and data submitted and run ‘as is’ on a suitable device
- **Data parallel Jobs**
  - MetaProcessor job consists of a task and multiple work units
  - Job provides an SPMD abstraction enabling data parallel operations
  - MGSI offers programming interfaces to develop application services
  - Application service is used to submit and manage MP jobs
  - User typically invokes the application service through a user interface
- **Redundancy**
  - Work units can be scheduled redundantly based on device availability
  - Redundant results can be compared for accuracy
- **Time Limits**
  - Job deadline applicable to the entire job
  - Workunit timeout specifies total resident time of a workunit on a device
  - CPU timeout specifies maximum run-time for a workunit on a device

back

Application Management

- **Registration**
  - Programs are registered as *tasks* in the system
  - Registered tasks may be shared by one or more applications
  - Tasks have platform-specific executables called *task modules*
- **Versioning**
  - Tasks and task modules have versions
  - Versions enable tasks to be centrally upgraded and managed
- **Phases**
  - Tasks can be released in *test*, *pilot* and *production* phases
  - Phases enable developers & administrators to unobtrusively release tasks

back

Application Development Environment

- **MetaProcessor Task Wrapper**
  - packages executable programs without source code modifications. Runs ‘as is’
  - Automatically includes features such as encryption and compression
  - Monitors and controls programs when executing on device
- **MetaProcessor Grid Services Interface (MGSI)**
  - Programmatic Web services interface based on HTTP, SOAP and XML
  - Support for 22 different programming languages
  - Minimal effort to develop *Application services*, tools and utilities
- **MetaProcessor Application Services**
  - Application services pre processes data, submit jobs and post processes results
  - Uses MGSI to interface with the MetaProcessor. Optionally can use batch utilities
- **MetaProcessor Task API**
  - Optional source code modifications for task check-pointing and monitoring
- **MetaProcessor SDK**
  - Rapidly migrate existing applications
  - Detailed documentation for all components
  - Reference implementation and value-added modules for rapid development

back

Data Management

- **Registration**
  - Data is registered as *work data* and *resident data* using MGSI
  - Data is uniquely named to avoid name collision
  - Data can be grouped as *work data set* and *resident data set*
- **Resident Data**
  - Data is read-only and cached on devices to reduce network traffic
  - Data may be shared by many jobs
  - Enables scheduling based on resident data affinity
- **Annotation**
  - Enables application services to store information such as, a data index

back

Product Scalability

Devices	Connections per hour, Data-In/hour	MP Dispatch Service	Management Service	Application Service	MP Database
500	**83 conn/hr, 4 MB/hr**	**1 Server – 1CPU, 1GB RAM, 50 GB disk, Linux**			
1,000	**167 conn/hr 8 MB/hr**	**1 Server – 1 CPU, 1GB RAM, 20 GB disk, Linux**			**1 Server-1cpu, 1GB RAM, 50 GB disk, Linux**
10,000	**1,667 conn/hr 83 MB/hr**	**1 Server-1cpu, 1 GB RAM, Linux**	**1 Server-1 cpu, 512 MB RAM, 30 GB disk (shared w/UD servers), Linux**		**1 Server-2cpu, 4 GB RAM, 100 GB disk, Linux**
25,000	**4,167 conn/hr 208 MB/hr**	**1 Server- 2 cpu, 1 GB RAM, Linux**	**1 Server-1cpu 512 MB RAM 100 GB shared disk, Linux**	**1 Server-1cpu 1 GB RAM, 20 GB disk, Linux**	**1 Server-2cpu 8 GB RAM, 150 GB disk. Linux + Warehousing**
150,000	**25,000 conn/hr 1,250 MB/hr**	**3 Servers-2 cpu, 1 GB RAM, Linux**	**1 Server-2 cpu 512 MB RAM, 500 GB shared disk, Linux**	**1 Server-1cpu, 1GB RAM, 20 GB disk, Linux**	**1 Server–4cpu, 8 GB RAM, 150 GB disk, AIX + Failover + Warehousing**

Manageability

- **Web-based administration Console**
  - Remote administration of users, devices, applications and jobs
  - Features accessible based on role and access control policies
  - System management functions to control core services
- **Self-updating infrastructure**
  - Automatic update of MP Agent and tasks
  - Phases and versioning enable smooth transition to new code
- **Automated systems management**
  - Periodic review and cleanup of stale data in the database and file system
  - Manager process restarts failed slave service processes
- **Rapid installation and upgrades**
  - Single command installation of MetaProcessor services in most environments
  - MP Agent compatible with most software distribution tools
  - Average time for MetaProcessor deployment is less than a day
  - Customized migration scripts to enable smooth upgrades

back

Security

- **Authentication**
  - User access to MetaProcessor requires an identifier and password
  - SSL-like protocol for authentication and network encryption key generation
  - Unique device identifiers and network session keys for authenticating devices
- **Encryption and Checksum**
  - Network communications encrypted using triple-DES network key
  - All data stored on devices is encrypted using triple-DES device key
  - All files stored on the devices are tamper-proofed using checksum validation
- **Organizations and Roles**
  - Users and applications allocated into organizations
  - User roles based on four levels of access control to the system
- **Digital Signature**
  - Application executable modules may be signed and validated on devices
  - DSA signature keys can be modified for each customer
- **Constrained execution environment**
  - MP Agent executes tasks in a sandbox with limited access to device resources

back

Device Unobtrusiveness

- **Preference profiles**
  - Control computation and communication time windows
  - Control disk space usage on devices
  - Control tasks that can execute on devices
- **Agent deployment**
  - Installation does not require a machine reboot
  - Deployed silently using enterprise software distribution tools
  - Run as a protected process such as, ‘WinNT Service’ or a user level application
- **Optimal usage of resources**
  - MP agent has a negligible memory foot-print
  - Lowest priority supported by the operating system
  - Network communication only after task completion
  - Limit disk usage based on device preference settings
  - Optionally configured to run in ‘screen saver only’ mode
- **Optional User control**
  - User can snooze the task on non-dedicated devices
  - User can shutdown MP agent

back

Platform Heterogeneity

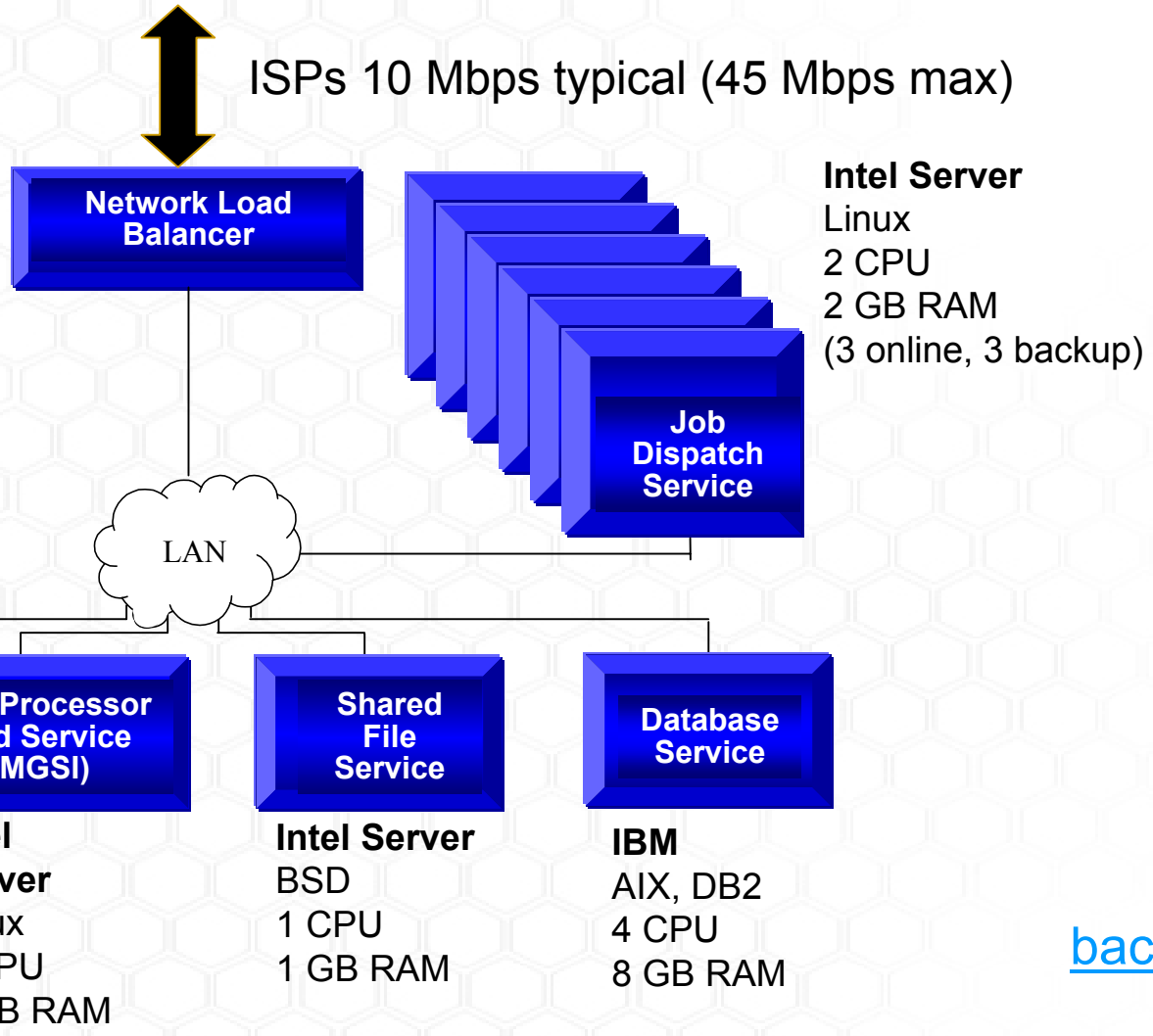
- **MetaProcessor Agent**
  - Windows 98, Window NT 4.0, Windows 2000 & Window XP
  - Linux Red Hat 7.2. Easily portable to other Unix versions
- **MetaProcessor Services**
  - Linux Red Hat 7.2. Portable to other Unix versions based on customer demand
  - MGSI accessible via 22 programming languages. Microsoft .NET-enabled
  - Highly portable Perl-based command line utilities

back

Highly Scalable Architecture – 1.5M Devices

Average
~150K unique devices/day
~300K results/day
~5K SQL statements/day

Peak
~300K unique devices/day
~1M results/day
~15M SQL statements/day



back