End-to-End Distributed Application Monitoring using the "Distributed Monitoring Framework"

Brian L. Tierney
(bltierney@lbl.gov)

Lawrence Berkeley National Laboratory

Background and Outline

- My Background:
  - >10 years experience working with data intensive distributed systems
    - remote visualization from a Cray-YMP: SC '91
    - Gigabit network testbed community (MAGIC)
    - Data Grid community
  - learned a lot about TCP and networking issues along the way
- This talk will cover:
  - Definition of the End-to-End problem
  - Components needed to solve this problem
  - Previous LBNL work on parts of the problem:
    - NetLogger and GMA
  - Planned work

Overview

- The Problem
  - When building distributed systems, we often observe unexpectedly low performance
    - the reasons for which are usually not obvious
  - The bottlenecks can be in any of the following components:
    - the applications
    - the operating systems
    - the disks, network adapters, bus, memory, etc. on either the sending or receiving host
    - the network switches and routers, and so on
- The Solution:
    - Highly instrumented systems with precision timing information and analysis tools

Performance Analysis

- Distributed system users and developers blame performance problems on network congestion
  - This is often not true!
- In our experience tuning distributed applications, performance problems are due to:
  - network problems: ~45%
    - this include TCP tuning issues
  - application design problems/bugs: ~45%
    - 50% client , 50% server
  - host / disk problems: ~10%
- Therefore it is equally important to instrument the applications

Solution

- A complete End-to-End monitoring framework that includes the following components:
  - instrumentation tools (application, middleware, and OS monitoring)
  - host and network sensors (host and network monitoring)
  - sensor management tools (sensor control system)
  - event publication service
  - event archive service
  - event analysis and visualization tools
  - **a common set of protocols for describing, exchanging and locating monitoring data**

Common Protocols

- We need a monitoring framework that provides a unifying view to a wide range of sensor data, from network to host to application.
- This requires common protocols and data formats:
  - event data descriptions
  - event dictionaries
  - query format
  - publish/subscribe APIs and protocols
  - timestamp format
  - types
  - etc.
- **Using XML-based solutions for this problem**
  - working with Global Grid Forum to define these (http://www.gridforum.org/)

Uses for Monitoring Data

- Monitoring Data not just for End-to-End Performance Analysis:
- Lots of “middleware services” need Monitoring data too:
  - Grid Schedulers
    - —find the best match of CPUs and data sets for a given job
  - Grid Replica Selection
    - —find the “best” copy of a data set to use
  - Reliable File copy service
    - —detect failures and recover
  - Network-aware applications
    - —TCP buffer size tuning, number of parallel streams, etc.

DMF

Instrumentation Tools

- Need to instrument applications, middleware, and operating systems
- Requirements:
  - non-intrusive
  - easy to use
  - real time monitoring ability
  - standard format(s)
  - accurate data
    - precision timestamps (see next slide)
    - indication of accuracy of the data (e.g.: “confidence level”)
- Issue: Application level vs. OS level monitoring
  - need application source code *OR*
  - need to intercept OS calls using shared library tricks *OR*
  - need to modify OS
  - Often hard to convince developers to instrument their code

DMF

Timestamps: Clock Synchronization Issues

- To correlate events from multiple systems requires synchronized clocks
- But how accurate does this synchronization need to be?
  - We have found that to analyze systems from the "user perspective" requires:
    - microsecond resolution between events on a single host (gettimeofday() system call)
    - millisecond resolution between WAN hosts
      - —fairly easy to achieve this with NTP
    - somewhere in between for LAN hosts
- Recommendation: everyone use IETF timestamp standard
  - example: 2002-01-18T21:20:07.401662Z
  - YYYY-MM-DDTHH:MM:SS.SZ (T=date-time separator, Z = GMT)
  - http://www.ietf.org/internet-drafts/draft-ietf-impp-datetime-05.txt

DMF

Host and Network Sensors

- Need a variety of host sensors
  - CPU, Disk, Memory, etc.
- Need non-intrusive network sensors capable of end-to-end **and** hop-by-hop network analysis
  - latency, capacity, available bandwidth, etc.
- Need standard schemas and publication mechanisms for this sensor data
  - SNMP only partially addresses this problem
    - Simple GET/SET model only
      - —**No support for subscription**
    - Source of information is implicit in the packet addressing
    - We can easily write tools to wrap SNMP data with our data format
    - SNMP is not well suited to application monitoring

DMF

Sensor Management

- As distributed systems become bigger and more complex, there are more pieces to monitor and manage
- Various components require different levels of monitoring:
  - constant
  - “on demand”
  - when triggered by some other event
- The sensors themselves need to be automatically installed, updated, and removed

- Requirement:
  - a Sensor Management System capable of securely controlling the distribution and execution of monitoring sensors in a distributed environment

DMF

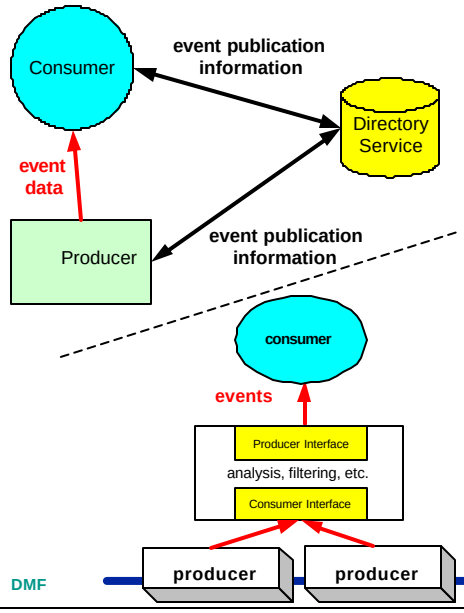
Event Publication

- To handle potentially huge amounts of event data requires an event publication and subscription service that is:
  - flexible
  - highly scalable
  - provides near real-time access to monitoring data
- The Global Grid Forum (GGF) (www.gridforum.org) has defined the “Grid Monitoring Architecture” (GMA), for this purpose.
  - Several GMA implementations have started to appear
- A great deal of work remains to define standard event schemas and event dictionaries for the GMA.

DMF

GMA Terminology and Architecture

- (Performance) Event:
  - Typed collection of data with a specific structure
- Producer Interface:
  - makes performance data (events) available
- Consumer Interface:
  - receives performance data (events)
- Directory Service:
  - supports information publication and discovery
  - must be distributed and/or replicated



DMF

Event Archives

- Archived event data is required for
  - performance analysis and tuning
    - compare current performance to previous results
  - accounting
- The archive must be extremely high performance and scalable to ensure that it does not become a bottleneck.
  - heavily loaded FTP server could generate about 500 KB/sec (1.8 GB/hr) of monitoring event data
  - e.g.: use pipelining to guarantee that applications and sensors never block when writing to the archive
    - buffer event data on disk
- SQL capability desirable
  - ability to do complex queries

DMF

Event Analysis and Visualization Tools

- Requirements
  - real time and post-mortem analysis capabilities
  - ability to correlate application events with host and network events
    - most existing tools do one or the other
  - flexible
  - configurable
  - etc.
- Tradeoff issues between flexibility and ease of use
  - special purpose, easy to use tools needed too

DMF

Existing Pieces

- Many of these components already exist or are in progress:
  - instrumentation tools
    - Pablo (UIUC), NetLogger (LBNL), log4j (apache), web100, etc.
  - host and network sensors
    - too many to list
  - sensor management tools
    - JAMM (LBNL)
  - event publication service
    - MDS (Globus), NWS (UCSB), R-GMA (RAL), CODE (NASA AMES)
  - event archive service
    - netarchd (LBNL), NWS (UCSB)
  - event analysis and visualization tools
    - lots, but most only work for specific types of events:
      - **NetLogger nlv (LBNL), Probe (Stazi), Autopilot (UIUC), etc.**
- **BUT, all use different event formats and protocols!**
  - **no interoperability**

DMF

New LBNL Project: The Distributed Monitoring Framework (DMF)

- The “Distributed Monitoring Framework” (DMF) Project will:
  - define common protocols and data formats
    - we are leading a GGF effort in this area
  - work with others to integrate existing components using this framework
    - e.g.: PingER, NWS, MDS
  - develop missing pieces
    - e.g.: event archives
- Goal:
  - provide a unifying view to a wide range of sensor data, from network to host to application

Existing Component: NetLogger

NetLogger Toolkit

- We have developed the NetLogger Toolkit (short for **Net**worked Application **Logger**), which includes:
  - tools to make it easy for distributed applications to log interesting events at every critical point
  - tools for host and network monitoring
  - event visualization tools that allow one to correlate application events with host/network events
- NetLogger combines network, host, and application-level monitoring to provide a complete view of the entire system.
- Open Source, available at http://www-didc.lbl.gov/NetLogger/

DMF

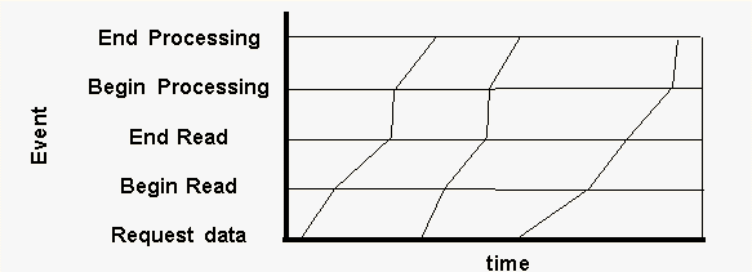
Sample NetLogger Use

```
import netlogger

nl = netlogger.NetLogger(
  “x-netlog://host.lbl.gov”,
  netlogger.NL_ENV )

while not done :
   nl.write("EVENT_BEGIN",“SIZE=%d“,(size,));
   done = do_something(data, size)
   nl.write ("EVENT_END","SIZE=%d“,(size,));

del nl
```

DMF

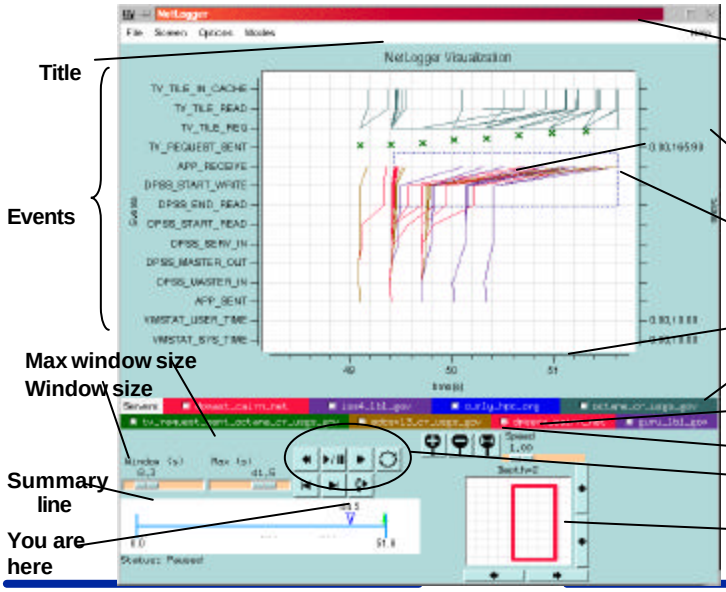
NetLogger Analysis: Key Concepts

- NetLogger visualization tools are based on time correlated **and** object correlated events.
  - precision timestamps (default = microsecond)
- If applications specify an "object ID" for related events, this allows the NetLogger visualization tools to generate an object "lifeline"
- In order to associate a group of events into a "lifeline", you must assign an "**Event ID**" to each NetLogger event
  - Sample Event ID: file name, block ID, frame ID, etc.



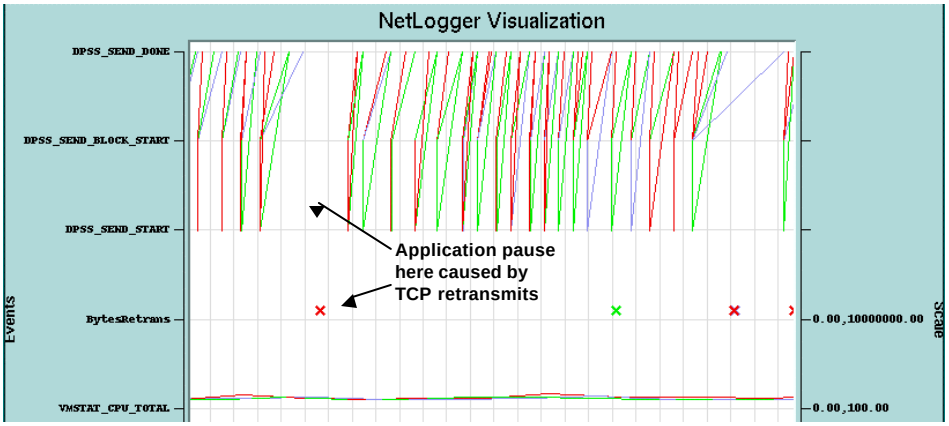
DMF

NLV Analysis Tool: Plots Time vs. Event Name



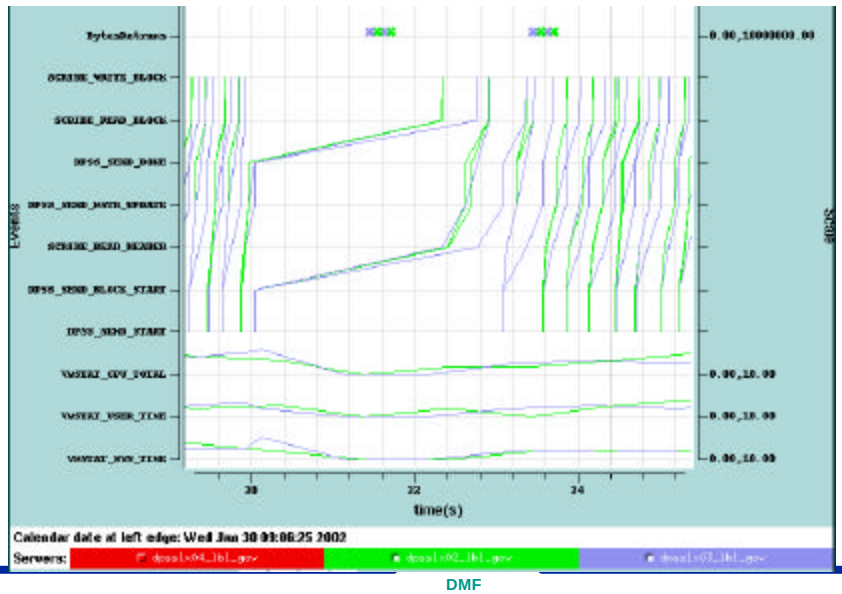
DMF

Example: Combined Host and Application Monitoring



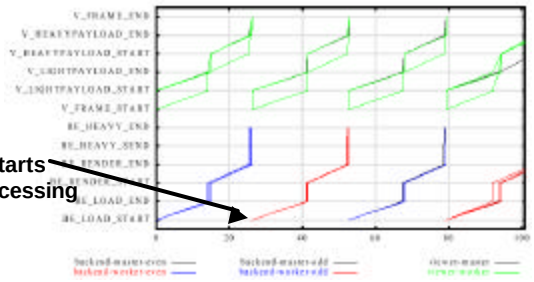
- 3 colors represent 3 parallel sockets
- X-axis = event; Y-axis = time
- Application Events: send header, send data start, send data end

Example: Combined Host and Application Monitoring

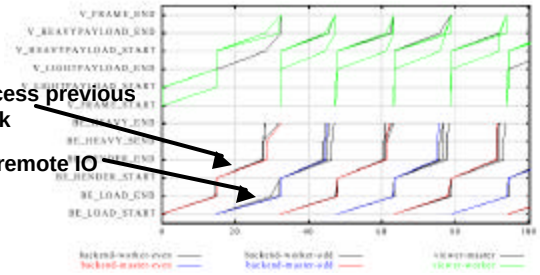


NetLogger Tuning Results

- I/O followed by processing



- overlapped I/O and processing



almost a 2:1 speedup

DMF Enhancements to NetLogger

- Rewrite of client library
  - Multiple log formats allowed with same API
    - ASCII (ULM)
    - Binary log format
      - **much better performance than ASCII**
      - **strong type information on the wire**
  - Other language APIs automatically generated with SWIG
    - Much faster than “100% native” implementations, esp. for script languages such as Perl, Python, and TCL
    - Changes and bug fixes in core automatically propagated to all APIs
- SOAP/WSDL descriptions
- Enhanced reliability
  - periodically try to reconnect broken TCP pipe
  - stores data on local disk while net is down

Role of Web Services in the DMF

DMF will be based on Web Services

- XML is the de-facto standard for language-independent and platform-independent self-describing data
- Web Services builds on XML (and HTTP) to provide:
  - a messaging protocol (SOAP)
  - an interface description language (WSDL)
  - a directory service (UDDI)
- The commercial sector is driving creation of Web Services tools and libraries
  - tremendous number of new tools emerging
- We can use these tools and libraries to speed up development of interoperable interfaces to Grid services, such as the GMA.

14

SOAP Example: Request for ping data

```
<SOAP-ENV:Envelope  … header stuff removed… <SOAP-ENV:Body>
 <gprq:Query> <gpe:Event>
    <Name xsi:type="xsd:string">Ping</Name>
    <gpe:ComplexElement SOAP-ENC:arrayType="EventItem[]">
        <gpe:SimpleElement>
            <Name xsi:type="xsd:string">SourceHost</Name>
            <Value xsi:type="xsd:string">foo.lbl.gov</Value>
        </gpe:SimpleElement>
        <gpe:SimpleElement>
            <Name xsi:type="xsd:string">DestHost</Name>
            <Value xsi:type="xsd:string">www.mit.edu</Value>
        </gpe:SimpleElement>
    </gpe:ComplexElement>
</gpe:Event> </gprq:Query>
```

DMF

SOAP Example: Reply for ping data

```
<SOAP-ENV:Envelope  … header stuff removed… <SOAP-ENV:Body>  <gpry:QueryReply>
 <gpe:Event>
    <Name xsi:type="xsd:string">Ping</Name>
    <TimeStamp>2002-01-18T21:20:07.401662Z</TimeStamp>
    <gpe:ComplexElement SOAP-ENC:arrayType="EventItem[]">
        <gpe:SimpleElement>
            <Name xsi:type="xsd:string">SourceHost</Name>
            <Value xsi:type="xsd:string">foo.lbl.gov</Value> </gpe:SimpleElement>
        <gpe:SimpleElement>
            <Name xsi:type="xsd:string">DestHost</Name>
            <Value xsi:type="xsd:string">www.mit.edu</Value> </gpe:SimpleElement>
        <gpe:SimpleElement>
            <Name xsi:type="xsd:string">RTT</Name>
            <Value xsi:type="xsd:decimal">68.300000</Value>
            <Units xsi:type="xsd:string">ms</Units> </gpe:SimpleElement>
        <gpe:SimpleElement>
            <Name xsi:type="xsd:string">Avg_10</Name>
            <Value xsi:type="xsd:decimal">60.100000</Value>
            <Units xsi:type="xsd:string">ms</Units> </gpe:SimpleElement>
    </gpe:ComplexElement> </gpe:Event>
</gpry:QueryReply>
```

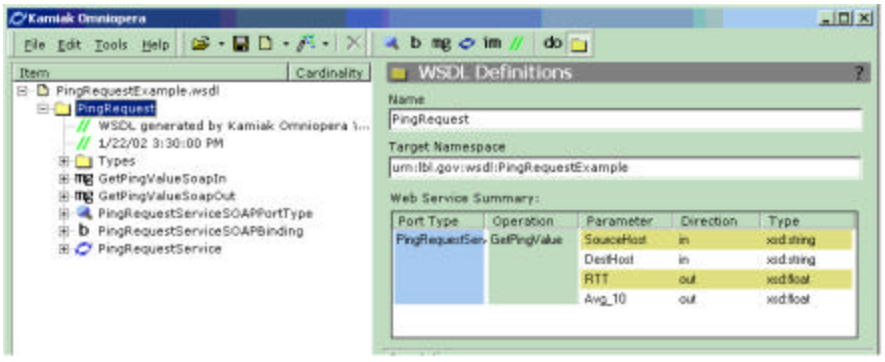
DMF

Web Services: WSDL

```
<wsdl:definitions name="PingRequest" xmlns:="...PingRequestExample " xmlns:xsd ...>
        <wsdl:types> <xsd:schema targetNamespace ="...PingRequestExample ">
                <xsd:complexType name=" GetPingValue_InParameters">
                        <xsd:sequence>
                                <xsd:element name=" SourceHost " type="xsd:string"/>
                                <xsd:element name="DestHost " type="xsd:string"/>
                        </xsd :sequence>
                </xsd :complexType >
                <xsd:complexType name=" GetPingValue_OutParameters">
                        <xsd:sequence>
                                <xsd:element name="RTT" type="xsd:float"/>
                                <xsd:element name="Avg_10" type="xsd:float"/>
                        </xsd :sequence>
                </xsd :complexType >
        </xsd :schema> </wsdl:types>
        <wsdl:message name="GetPingValueSoapIn">
                <wsdl:part name="Parameters" type=" GetPingValue_InParameters"/>
        </wsdl:message>
        <wsdl:message name="GetPingValueSoapOut ">
                <wsdl:part name="Parameters" type=" GetPingValue_OutParameters"/>
        </wsdl:message>
        <wsdl:portType name=" PingRequestServiceSOAPPortType ">
                <wsdl:operation name=" GetPingValue">
                <wsdl:input name="GetPingValueInput" message="GetPingValueSoapIn "/>
                <wsdl:output name="GetPingValueOutput " message="GetPingValueSoapOut/>
                </wsdl:operation>
        </wsdl:portType >
<wsdl:binding name="PingRequestServiceSOAPBinding"          type=" PingRequestServiceSOAPPortType ">
        <soap:binding style=" rpc " transport="../soap/http"/>
        <wsdl:operation name=" GetPingValue"> <soap:operation style="rpc "          soapAction="urn:lbl.gov:wsdl:PingRequestExample /GetPingValue"/>
            <wsdl:input name="GetPingValueInput">
                <soap:body use="encoded" encodingStyle="...soap/encoding/"/>
            </wsdl:input>
            <wsdl:output name="GetPingValueOutput">
                        <soap:body use="encoded" encodingStyle="..soap/encoding/"/>
                </wsdl:output>
        </wsdl:operation>
</wsdl:binding>
<wsdl:service name="PingRequestService">
                <wsdl:port name="PingRequestServiceSOAPPort "                                                    binding=" PingRequestServiceSOAPBinding">
                <soap:address location="http://www.lbl.gov/DIDC/PingRequest "/>
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```


WSDL Tools

- Generating all this is much easier than it looks.
- There are tools to simplify the process:
- GUI for authoring and editing web service contracts



For More Information

DMF: http://www-didc.lbl.gov/DMF/

GMA: http://www-didc.lbl.gov/GGF-PERF/GMA-WG/

email: bltierney@lbl.gov