MRNet for Scalable Tool Communication and Data Analysis

Dorian Arnold
Paradyn Project

CScADS 2008 Workshops
July 21 – 24, 2008
Snowbird, UT

Paradyn

Goals (This Week)

- Identify necessary components for highly scalable tools
- Present MRNet component model for scalable tools
- Identify how MRNet model can be extended to further support the needs of scalable tools

Tool Front-end

Result Presentation

Scalable Communication Infrastructure

Scalabl

Tolerance

Gro

MRNet

lable Analysis

Topology Management

Tool Back-end

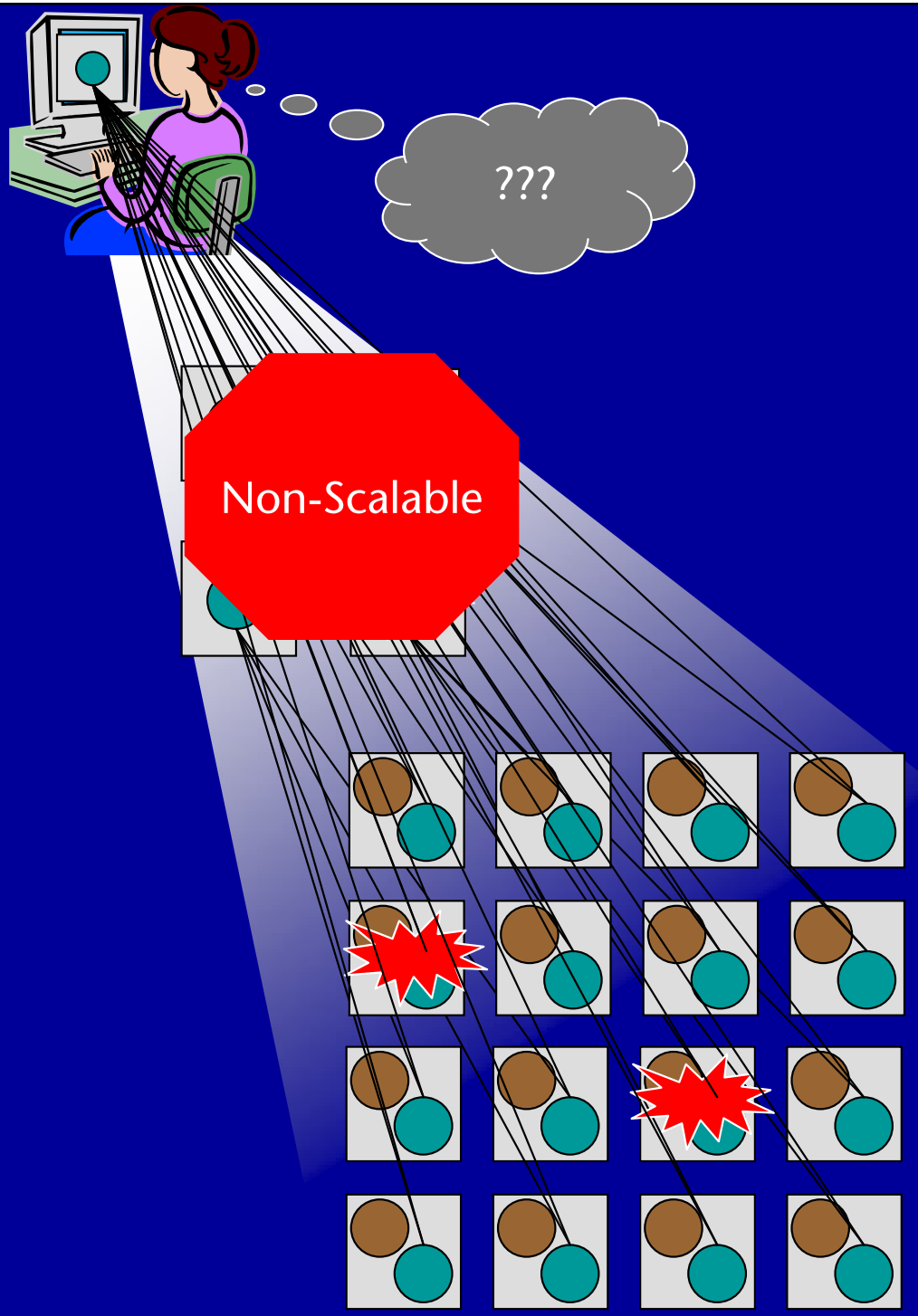
Profile, Monitor, Control

Application

Custom Network Interfaces

Resource Management

Hardware Resources



MRNet

- C++ library linked into tool front-end and back-ends
- "comm_node" program implements tree-based overlay network (TBŌN)
- No root privileges necessary
- Actively maintained implementation
  - v2.0 released 07/2008
  - Open-source
  - LGPL license
  - http://www.paradyn.org/mrnet
  - BG/L, HPC clusters
    - x86, ia64, powerpc64
    - Linux, AIX, Solaris, Windows

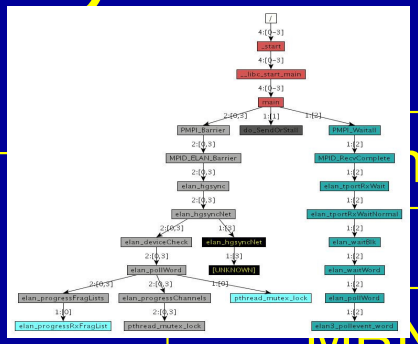
MRNet Component Model

- Demonstrate using the MRNet-based Stack Trace Analysis Tool (STAT)
- LLNL collaborators: Dong Ahn, Bronis de Supinski, Greg Lee, Martin Schulz @ LLNL
- Use stack trace samples over time to identify process equivalence classes
  - Reduces exploration space from $O(10^5)$ to $O(10^1)$
  - Complements full-featured debuggers

Stack Trace Analysis Tool

Stack Trace Analysis Tool

trace(count, freq.)



FE

nt-end

CP

MRNet Communication Process

STAT Filter

STAT Daemon

Application Processes

BE

MPI

MRNet: Topology Management

```
node0:0 =>
     node1:0
     node2:0;

node1:0 =>
     node3:0
     node3:1;

node2:0 =>
     node3:2
     node3:3;
```

- Topology generation tools
  - Balanced trees, e.g. "16^2"
  - Specification for arbitrary trees
  - Do not consider system characteristics
- Users can provide their own topologies
- Dynamic topologies
  - Reconfiguration due to process/node failures
  - Self monitoring, self optimizing
    - Reconfigure due to performance failures
- Factor system characteristics
  - Hardware topology
  - Performance models & measurements

STAT Front-end

```
main(){
    Network * net = new Network (topology_file);

    Communicator * comm = net->get_BroadcastComm();

    Stream * stream =
        new Stream( comm, STAT_FILTER, WAITFORALL);

    stream->send( PROT_SAMPLE_TRACES, "%d %d",
                  num_samples, sampling_freq );

    stream->recv( &tag, &packet);

    packet->unpack( "%ac", &call_graph_byte_array );

    //post-processing: export .jpg of call graph
}
```

MRNet: Scalable Communication Component

- Front-end to back-end communication
  - Blocking or non-blocking
  - Point-to-point
  - Multicast
  - Gather
  - Scatter
- Abstract communication interface
- Inter-daemon communication?

STAT Front-end

```
main(){
    Network * net = new Network (topology_file);

    Communicator * comm = net->get_BroadcastComm();

    Stream * stream =
        new Stream( comm, STAT_FILTER, WAITFORALL);

    stream->send( PROT_SAMPLE_TRACES, "%d %d",
                  num_samples, sampling_freq );

    stream->recv( &tag, &packet);

    packet->unpack( "%ac", &call_graph_byte_array );

    export_CallGraphToJPEG( call_graph_byte_array );
}
```

STAT Back-end

```
main(){
    Network * net = new Network ( … );

    net->recv( &tag, &packet, &stream );

    packet->unpack( "%d %d",
                    &num_samples, &sample_freq );

    //uses StackWalker API component
    local_callgraph =
        collect_Samples( num_samples, sample_freq );

    stream->send( PROT_CALLGRAPH, "%ac",
                  local_callgraph );
}
```

MRNet: Scalable Analysis Component

- *Wait For All*, *Time Out*, *Don't Wait* Synchronization
- Built-in, general purpose filters
  - Min, max, sum, count, average, concatenate
- Dynamically loadable tool-specific filters:

```
load_Filter( const char * inFilterFunctionName,
             const char * inFilterSharedObject );
```

- Composable filters
- Community filter repository?
- Different filters @ different levels?

STAT Filter

```
filter( vector<Packet> inPackets,
        vector<Packet> outPacketsToParent,
        vector<Packet> outPacketsToChildren,
        Packet inFilterParameters,
        void ** inoutFilterStorage ){

    for( i=0; i<inPackets.size; i++){
        trace = deserialize( inPackets[i] );
        ret_trace = merge( ret_trace, trace );
    }

  Packet p = serialize( ret_trace );

  outPacketsToParent.pushback(p);
}
```

MRNet: Fault-Tolerance

- Basic failure detection mechanisms
- Failures → tree reconfiguration
- Recover filter state (for certain classes)
  - Potentially missing output for others

What next?
How to extend the MRNet model?

- Communication Model
  - Direct point-to-point
  - Inter-daemon communication
- Aggregation Model
  - Different filters @ different levels
- Other suggestions/feature requests
  - Different programming models
    - E.g. map/reduce, group RPC, declarative
  - Language bindings
  - …

Backup Slides

MRNet in Action

- CEPBA-Tools (Universitat Politecnica de Catalunya)
- OpenlSpeedShop (Krell Institute)
- Paradyn (University of Wisconsin)
- Stack Trace Analysis Tool (LLNL)
- TauOverMRNet (University of Oregon)
- TBON-FS (University of Wisconsin)

- Initiating collaborations with
  - TotalView, RENCI/UNC, Jülich Supercomputing, …

MRNet Success Story: STAT on Blue Gene/L