GridStat: A Status Dissemination Middleware for Critical Infrastructures

Harald Gjermundrød
Talk Outline

- Background and Motivation
- GridStat Framework
- Adaptive Mechanism
- Pattern Mechanism
- RPC Mechanism
- Conclusion
Wide area control and monitoring application for critical infrastructures such as those for the electric power Grid are distributed by nature.
Background (US Electric Power Grids)
Background (Cont.)

- Higher demand for power transmission – miles x megawatts
  - More power and longer distances with little new transmission capacity
- More participants whose actions affect grid stability
- Technology in recent years is adding
  - Many more “intelligent” devices
  - Much more heterogeneity
- Protection and control is mostly local today
  - Remedial Action Schemes (RAS): hardwired remote link to trigger a protective relay
  - Otherwise almost exclusively local monitoring (status) & local control
    - Power dynamics are grid wide, and anomalies can affect a wide geographic area
Motivation

- Some properties that can be observed from these systems
  - System must be **closely monitored**; demand must always be matched by the supply
  - **Tightly controlled**, i.e. complete control over the network resources
  - **Semantic of the information is known**, i.e. what type of data, how often it is pushed into the network, and who and where are the sinks located
  - **Slowly changing**, i.e. the information flow stays fairly static
Motivation (Cont.)

- Middleware provides
  - Higher abstraction for the developers to build distributed applications

- Application logic provided as a **middleware service**
  - **Reuse** of application logic
  - **Easier** to develop application logic
  - **Reduced** use of computational and network resources

- Adaptability of information flows
  - **Emergency situations**, like failure of power lines, power-plants, etc
  - Seasonal changes or bad weather forecast
  - Scheduled maintenance of major power lines, power-plants, etc

- Modify control activators
  - Need for **mechanisms** to control and manage the settings of **activators**
  - One way pub-sub communication not adequate for this
GridStat Framework

- GridStat is a wide-area publish/subscribe middleware developed for disseminating streams of status information for the electric power grid
  - Optimized for the domain of critical infrastructures, which makes it possible to take advantage of the semantics of the status information
  - Convey status data in a reliable, timely and secure manner (QoS)
  - Also applicable to status dissemination needs of other infrastructures: transportation, water, gas, etc
GridStat Framework (Cont.)

- Middleware provides for a mechanism to manipulate the raw event streams in order to **produce new event stream** at the middleware level.

- Middleware provides a mechanism for **rapidly changing subscription sets**
  - Two algorithms provided with different tradeoffs

- Middleware provides a mechanism for advanced **RPC with built-in features** for:
  - Fault tolerance using three redundancy techniques
  - Safety through Pre-and Post conditions
GridStat is Publish-Subscribe Middleware

- Publish-subscribe architecture
  - Publish: periodically announce status values
  - Subscribe: periodically receive status values
  - Simple, CORBA-compliant APIs for both publishers and subscribers, management/control infrastructure, etc.
  - Subscribers have transparent cache of latest status value
  - Network of internal servers managed for QoS
  - Optimized for semantics of status items
    - Not just arbitrary event delivery like generic publish-subscribe

![Diagram of Publisher and Subscribers]
GridStat Functionality

- **Management plane**: controls the resources in the data-plane
- **Data plane**: forwards events with *end-to-end QoS*

**GS Management Plane**

- **QoS Meta-Data**
- **QoS Control**
- **QoS Requirements**

- **Generator**
- **PMU**
- **Area Controller**

**Wide Area Computer Network (GS Data Plane)**

**Publishers**

**Subscribers**

EPL671: Research Methods in Computer Science at University of Cyprus 05/03/2010
GridStat Detailed Architecture

- Management plane is a hierarchy
- Data plane consists of a set of clouds
GridStat Subscriber APIs

- **Pull / Pull Set**
  - A *cache instance* of the variable kept at each subscribe
  - Subscriber can use just like a *local object*, when needed
  - Distribution *transparency*
  - *Data-point set* with same timestamp, these can be *Pulled as a set*

- **Push / Push set**
  - Subscriber can register to get each update through a *callback object*
  - Good for database integration and/or visualization

- **QoS Push**
  - Subscriber can register *callback* to get notified if *QoS violated*
  - Most apps won’t use, but great for logging: end-to-end QoS violation
Route Allocation to Subscriber 1
Route Allocation to Subscriber 2

Note: Sub2 may have a different rate, latency, or redundancy than Sub1
**A Crucial Note on Network & Publisher Load**

Direct network programming: up to 37 copies (#subs) of a given update:

![Diagram of network and publisher load]

Problems: (1) network load (2) publisher load (3) multiple encrypts

Note on IP Multicast (1) not everywhere (2) can’t do per-sub QoS

GridStat: 1 copy (max) of a given update on any network link:

![Diagram of IP Multicast]

Note: per-subscriber QoS (rate, latency, #paths) via rate filtering: if a subscriber (or subtree) does not need a given update it is not sent.
How GridStat Can Help the Grid

- Better situational awareness for operators of emerging problems
- Better reliability of grid
- Many more control and protection schemes possible, and cheaper
  - Much of today’s costs for Specialized Protection Schemes (SPS) are the hard-coded, “one-off” data communications
- Increased efficiency “exciting” new potential for better monitoring (IEEE, 2006)
- Better cyber-security
- Advanced Metering Infr. (AMI)/Demand Response (DR)
- “Smart Grids require smart data: flexible, robust, & secure”!
Talk Outline

- Background and Motivation
- GridStat Framework
- Adaptive Mechanism
  - Mode Change Mechanism and Management
    - Hierarchical Mode Change Algorithm
    - Flooding Mode Change Algorithm
  - Experimental Results
    - Completion Time
    - Scalability
- Pattern Mechanism
- RPC Mechanism
- Conclusion
Adaptive Mechanism

- Results published in the following paper:

Mode Change Mech. and Management

- **QoS Hierarchy**
  - A *mode* is a named *set of subscriptions* and owned by a QoS Broker
  - A QoS Broker owns a set of modes, but **only one mode is active** at any given time

- **Data Plane**
  - A Forwarding Engine have a *set of active* operating modes
  - A Forwarding Engine have as many *simultaneous active operational modes* as it have ancestors in the QoS hierarchy
  - A Forwarding Engine *forward events* if one of its *active operational modes* have a subscription to this event
Mode Change Operations

- A mode change is **initiated** by the QoS Broker which owns the mode

- QoS Broker commands Forwarding Engines in the Data plane to **change its routing table** to the new mode

- Two possible outcomes of a mode change
  - **consistent mode change**: all the routing tables are switch within the given time
  - **inconsistent mode change**: one or more of the Forwarding Engines have not changed the routing tables
    - Recovery mechanism are initiated in order to eventually reach a consistent mode change
Hierarchical Mode Change Algorithm

- All commands are propagated down through the hierarchy until the Leaf QoS Broker.
- Leaf QoS Broker commands all the FE in its cloud and waits for ACK from all of them; a single ACK is then propagated up the hierarchy again.
- Divided into five phases:
  - Inform Phase: Subscribers are informed about the change.
  - Prepare Phase: Edge FE creates a intersection (current ∧ next) routing table.
  - Internal Change Phase: Internal FE changes to new mode.
  - Edge Change Phase: All edge FE changes to the new mode.
  - Commit Phase: Subscribers informed about the change.
Hierarchical Mode Change Algo. Example

Change from Green to Yellow

Management Plane

Phase 1a: Inform sub
Phase 1b: Inform sub
Phase 1c: Inform sub
Phase 1d: ACK Inform sub
Phase 1e: ACK Inform sub

Phase 2a: Prepare
Phase 2b: Prepare Edge
Phase 2c: Prepare Edge
Phase 2d: ACK Prepare Edge
Phase 2e: ACK Prepare Edge

Phase 3a: Internal ch.
Phase 3b: Internal ch.
Phase 3c: Internal ch.
Phase 3d: ACK Internal ch.
Phase 3e: ACK Internal ch.

Phase 4a: Edge ch.
Phase 4b: Edge ch.
Phase 4c: Edge ch.
Phase 4d: ACK Edge ch.
Phase 4e: ACK Edge ch.

Phase 5a: Commit
Phase 5b: Commit
Phase 5c: Inform sub
Phase 5d: ACK Commit
Phase 5e: ACK Commit

Data Plane

Publisher
Subscriber

B1
B2
B3
B4
C1
C2
C3
C4

QoS Broker A
QoS Broker B
QoS Broker C

Modes: Stable, Unstable
Operates in: Stable

Modes: Green, Yellow, Red
Operates in: Green

Modes: Normal, Warning, Critical
Operates in: Normal

B* forwarding engines operate in Green and Stable
C* forwarding engines operate in Green and Normal
Flooding Mode Change Algorithm

- GridStat provides a limited flooding mechanism to flood an event to a set of clouds

- Flooding mode change algorithm uses this by embedding a publisher with the QoS Brokers

- QoS Brokers initiates a new mode in the following way:
  - Flood the cloud(s) to change a mode at a future time-stamp
  - Upon receiving a mode-change event FE ACKs to their Leaf QoS Broker
  - At the destined future time-stamp the FE activates the new mode
Flooding Mode Change Algo. Example

Phase 1a: Publish mode change

Phase 1b: Flooded

Phase 1c: Flooded

Phase 1d: Flooded

Phase 1e: Flooded

Phase 2a: All FE ACK

Phase 2b: All FE ACK

Phase 3: At time t all FE change to Yellow
Experimental Results

- The setup for the experiments
  - Three layers
  - Four clouds
Completion Time

- **Completion** of a mode change
  - Top Hierarchal (100 oper.)
  - Bottom Flooding (300 oper.)

- **Graphs**
  - Link delay: 0, 1, 2, 4, 8 ms
  - y-axis: average completion time in ms
  - x-axis: “reliability of com. Links”
    - [ Overall packet loss(%) : ]
    - Min consecutive packet loss :
    - Max consecutive packet loss ]
Scalability

- **Scalability**
  - Increase hierarchical scope for the hierarchical algorithm
  - Increase the flooding domain for the flooding algorithm

- **Graphs**
  - Change from level: top, middle, leaf
  - Link delay: 8 ms
  - y-axis: average completion time in ms
  - x-axis: “reliability of com. Links”
    - [ Overall packet loss(%) : ]
    - Min consecutive packet loss :
    - Max consecutive packet loss ]
Talk Outline

- Background and Motivation
- GridStat Framework
- Adaptive Mechanism
- Pattern Mechanism
  - Introduction to the Mechanism
    - Overview of the Condensation Function
    - Benefits of the Condensation Function
    - Design of the Condensation Function
    - Defining the Condensation Function
  - Experimental Results
- RPC Mechanism
- Conclusion
Pattern Mechanism

- Results published in the following paper:

Condensation Function Mechanism

- Condensation functions allow applications to define new derived status variables
  - Sometimes subscribers just read a large set of status items once to calculate a derived variable
  - Supported by allowing user-defined condensation functions to be loaded in status routers
  - Building block for other mechanisms/capabilities
- Can be dynamically loaded into FEs
Benefits of the Condensation Function

- Some benefits of the condensation function
  - Conservation of network resource
    - If it's placed close to the source
    - Reuse of application logic at the application layer
    - Logic only produces an output in rare cases
  - Conservation of computation resources
    - Reuse of application logic at the application layer
    - Logic only produces an output in rare cases
Design of the Condensation Function Mech.

- **Consists of the following 4 modules**
  - **Input filter** [optional]: filter status update events by value range
  - **Trigger** which initiates calculation:
    - Time triggered: every $x$ ms the calculator is triggered
    - Event triggered: received update events from $x$ input variables
    - Alert triggered: received alerts from $x$ of the subscribed input alert variables
    - User defined: user can supply the triggering object
  - **Calculator**
    - Init method: initializes the data structures
    - Calculation method: performs the aggregation of the events received
  - **Output filter** [optional]: like input filter

![Diagram showing the flow of data through the modules: Input Filter, Input Trigger, Calculator, Output Filter.]

- Filter the input data to either a lower or upper threshold
- When should the calculation start, delay or # of input
- Applies a user defined function to the arguments
- Filter the result to either a lower or upper threshold
Design of the Condensation Fun. Mech. (Cont.)

- The condensation function **implements** the same **interface** as a **communication link**

- I.e. the condensation function **doesn’t interfere** with the forwarding of events
Defining a Condensation Function

- **Creation with GUI-based tool**
  - Specify input and/or output filter
  - Specify input variables
  - Specify triggering method
  - Specify calculator object

- The produced status variable is registered just like a new publication

- Placed in cloud with input variables (present limitation)
Purpose of experiments to show the **resources savages** of placing application logic into the middleware layer.
Condensation Function Definition

- **Published variables**
  - Pub \( p_0, p_1 \): 1 variable each published every 50 ms
  - Pub \( p_2, p_3 \): 20, 40, 60 variable each published every 1 ms
  - Pub \( p_3 \): 1 alert variable published every 100 ms

- **Subscribed to variables**
  - Sub \( s_0 \): Variable published by \( p_0 \)
  - Sub \( s_1, s_2 \): Variable published by \( p_1 \)
  - Sub \( s_3, s_4 \): All variables published by \( p_2 \) and \( p_3 \)

- **Application logic which is performed at \( p_2 \) and \( p_3 \)**
  - Once an alert is detected from \( p_3 \) calculate the average of all variables published by \( p_2 \) and \( p_3 \) and publish this average
Experimental Results End-to-End Latency

- Pub $p_1$ – Sub $s_1$
  - Top without condensation function
  - Bottom with condensation function
- Graphs
  - Y-axis: % of the total events received
  - X-axis: latency of events
- An improvement of about 0.2 ms
Experimental Results End-to-End Latency

- Pub \( p_1 \) – Sub \( s_2 \)
  - Top without condensation function
  - Bottom with

- Graphs
  - Y-axis: % of the total events received
  - X-axis: latency of events

- Big improvement for the high load patterns
Experimental Results

- **Mean and SD**
  - Small improvement to the mean and SD for Sub s₀ and Sub s₁, but large improvement for Sub s₂

- **CPU usage**
  - i₀ about 10% reduction when condensation function is loaded
  - i₁ and i₂ huge load reduction
Talk Outline

- Background and Motivation
- GridStat Framework
- Adaptive Mechanism
- Pattern Mechanism
- RPC Mechanism
  - Ratatoskr RPC Mechanism
    - Architecture
    - Timeliness
    - Redundancy
    - Safety
  - Evaluation
    - Temporal Redundancy over Omission Faults Rates
    - Spatial Redundancy over Transient Faults
- Conclusion
RPC Mechanism

- Results published in the following paper:

Ratatoskr RPC Mechanism

- **RPC mechanism** targeted to set *actuators* in the Electric Power Grid, with *tunable QoS* for
  - Timeliness
  - Redundancy
  - Safety
- Uses a **two-tiered architecture** on top of a QoS managed publish-subscribe middleware

**Tier 1**
- Provides a **2 Way over Publish-Subscribe (2WoPS)** communication protocol

**Tier 2**
- Provides RPC client and server functionality
Ratatoskr Architecture

Control center
- Control center system
  - Ratatoskr RPC
  - Legacy control
  - Sensor state monitor
- 2WoPS
- GridStat Publisher
- GridStat Subscriber

Substation
- Actuator
  - Substation sensors
  - Ratatoskr RPC
  - Legacy control
  - Protection scheme
- 2WoPS
- GridStat Publisher
- GridStat Subscriber

Application Layer

Network: UDP-IP, ATM, network processors, ...

RPC Client
- "Client" side of the Application

RPC Server
- "Server" side of the Application

QoS Managed Pub-Sub Middleware

Pub-Proxy & Sub Proxy

Underlying Network
Timeliness

- Ratatoskr RPC mechanism requires an underlying QoS managed Pub-Sub middleware

- One of the QoS dimension is end-to-end delay of a message delivery, hence timeliness

- The timeliness guarantees that the Pub-Sub middleware provides are exposed to the RPC call interface
  - User specifies the desired and required end-to-end delay when setting up the RPC “connection”
Redundancy

- Three Redundancy Techniques provided
  - Spatial Redundancy
    - How many disjoint network paths should the messages be sent over
  - Temporal Redundancy
    - How many copies of the same message should be sent and with what delay between each send
  - ACK/Resend
    - Require an explicit ACK to be sent back to the sender
    - Can specify how many resends of the message should be done if no ACK is received

- The user can combine any of the techniques as needed by the application
  - Use Spatial and Temporal for time delay sensitive application
  - Use ACK/Resend for non time critical application
Safety

- In some large scale infrastructures the client may not have the latest “state/view” of the condition
  - If an actuator is set wrongly serious consequences can occur
- Similarly the effect of setting an actuator may not be what was intended
- Pre- and Post conditions are built into the call semantic
  - **Pre-conditions** are conditional expressions that are evaluated at the server side before the execution of the RPC call
    - In case the condition evaluates to false, call is aborted
  - **Post-condition** are conditional expressions that are evaluated after (delay user defined) the RPC call is executed
    - In case the condition evaluates to false, this is reported back to the application through an exception
Evaluation

- The setup for the experiments
  - One QoS management layer
  - Two entry-point Forwarding Engines
  - Eleven Forwarding Engines which provide two paths from the Client to the Server
- Two fault models
  - Omission faults with uniform drop probability
  - Transient periods of total link failures
Temporal Redundancy over Omission Faults

- Omission Fault Rates
  - 1%, 2%, 4%, and 8%

- Temporal Redundancy
  - 1, 2, 4, and 8 sends

- Experimental results match the predicted

- Results achieved
  - 1% drop 2 sends => 99.2% early success
  - 4% drop 4 sends => 99.92% early success
  - 8% drop 8 sends => 99.92% early success
Spatial Redundancy over Transient Faults

- **Transient Periods**
  - 1 sec. failure every $y$ sec
  - $y$ is: 10000, 1000, 500, 100, 50

- **Spatial Redundancy**
  - Two paths are set up
  - Predicted results by
    - 1s/50s = 2% loss
    - 1s/100s = 1% loss

- **Results achieved**
  - 1% drop (2 sends due to redundant paths) => 99.4% early success
End-to-End Latency With Standard Deviation

- **Omission Fault Rates**
  - 1%

- **Transient Periods**
  - 1 sec / 10000 sec

- **Temporal Redundancy**
  - 1 and 4

- **Spatial Redundancy**
  - One and off

- **Conclusions**
  - All requests are eventually satisfied
  - SD and end-to-end latency increases when temporal redundancy is used for fault tolerance
Conclusion

- GridStat: status dissemination middleware tailored for critical infrastructures specifically the electric power grid
  - Publish-subscribe architecture
  - Subscribers have transparent cache of latest status value
  - Network of internal servers managed for QoS
    - Timeliness
    - Redundancy
    - Security
Conclusion

- **Pattern mechanism**
  - Migrating application logic into the middleware layer
  - Provides the following benefits
    - Resource conservation
    - Reuse of components
    - Ease of developing the application logic

- **Adaptive mechanism**
  - Two mode change algorithms present in the GridStat framework
    - Hierarchical mode change algorithm enables subscriptions in both modes to flow during the transition (strong guarantees), but is “slow”
    - Flooding mode change algorithm is efficient, but provides best-effort guarantees

- **RPC mechanism**
  - Ratatoskr RPC mechanism over pub-sub middleware
    - Tailored for setting actuators in critical infrastructures
    - Provides tunable QoS to satisfy various applications
      - Redundancy
      - Timeliness
      - Safety
Future/Ongoing Work

- Vision: “The vision of the North American SynchroPhasor Initiative (NASPI) is to improve power system reliability through wide-area measurement, monitoring and control.”
  - Synchrophasor: a sensor with a very accurate GPS clock
  - Becoming much more deployed in US, Europe, …

- Great need for much better data delivery services
  - Can no longer send “all data to control center at the highest rate anyone might want to”

- GridStat team involved with spec of “NASPInet” services
  - Many requirements come from GridStat research (cited)

- Will be involved in a major wide-area pilot project starting in 2010 (some federal fiscal stimulus projects in late 2009)
GridStat Team

- **Faculty:** Dave Bakken, Carl Hauser, Anjan Bose
- **Current Students:**
  - Graduate: Wendy Maiden (PNNL), Leif Carlsen (PNNL), others soon
  - Undergraduate: Loren Hoffman, Dave Anderson, Nathan Schubkegel, Chad Selph
- **Alumni:**
  - Graduate students: Kim Swenson (MS 2009), Rasika Mudumbai (MS 2009), Sunil Muthuswamy (MS 2008), Erik Solum (MS 2007), Stian Abselsen (MS 2007), Erlend Viddal (MS 2007), Joel Hekley (MS 2007), Ioanna Dionysiou (PhD 2006), Harald Gjermundrød (PhD 2006), Venkata Irava (PhD 2006), Ryan Johnston (MS 2005), Suprith Sheshadri (MS 2005), Ping Jiang (MS 2004)
  - Undergraduates: about a dozen on senior projects (Avista)
Thank you for your attention!

Questions?

More info contact me at:
harald@unic.ac.cy

http://www.gridstat.net
http://www.cs.unic.ac.cy/harald