GridStat: A Status Dissemination Middleware for Critical Infrastructures

Presented by: Harald Gjermundrød
In collaboration with: David Bakken and Carl Hauser

Talk Outline
- Background and Motivation
- GridStat Framework
- Multicast with Temporal Rate Filtering
- Adaptive Mechanism
- Pattern Mechanism
- RPC Mechanism
- Conclusion

Background
- Wide area control and monitoring application for critical infrastructures such as those for the electric power Grid are distributed by nature

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
- Ability to take snap-shot of the state of the system
  - In order to run power estimation/stability functions
- Application logic provided as a middleware service
  - Reuse of 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
- 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 wide-area multicast with deterministic rate filtering provides the ability to take global snapshots
- 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
GridStat Architecture

Management plane is a hierarchy

Data plane consists of a set of clouds

Multicast W/ Filtering Mechanism

Results published in the following paper:

Multicast with Temporal Rate Filtering

- The routing table is organized in such a way that it provides:
  - 1) **Multicast**: only one copy of an event is sent if multiple paths have been allocated over the same link.
  - 2) **Deterministic Filtering**: the event is only sent out on a link if any of the subscribers for this event triggers it to be forwarded.

- Mechanism is placed on the (edge) forwarding engines (E)FE
  - No need for any management from the management plan
  - Transparent to the end-points
  - Always activated

- The multicast feature is a side-effect of the data-structure
  - No extra computational resources, but storage resources needed

- The filtering feature does use computational resources for each event that is forwarded

Assumptions for the Routing Mechanism

- The publishers will publish status events at the registered interval and they are using GPS clocks

- The subscribers only request the status event from the first phase of a subscription interval.
  - For example, a status variable is published every 10 ms and subscribed to every 30 ms.
  - Events with timestamp 0, 30, 60, 90, ... (first phase) are forwarded
  - Events from the two other phases (10, 40, 70, 100, ... and 20, 50, 80, 110, ...) are dropped

- If the requested subscription interval is not a multiple of the publishing interval, then the subscription interval will be converted to the closest (but always less than) multiple of the publishing interval

Temporal related status variables

- **Temporally related status variables**
  - **Status variables** that are published at the same rate
  - Events from the variables carry a GPS time-stamp
  - A set of variables are temporally related if they publish their events at the “same” GPS time

- **Deterministic temporally rate filtering**
  - Forwards events from different publishers with the same timestamp at the rate specified by the subscribers. The subscribers get the measurements that are taken at the same time from multiple places, while dropping the “right” events.
  - An example of how the filtering would be performed by a FE
    - Variables published at an interval of 10 ms
    - Two subscription paths with an interval of 20 ms and 50 ms
    - Event forwarded at time: 20, 40, 50, 60, 80, 100, 120, 140, 150, 160, ...

Temporal Rate Filtering Algorithm

- The filtering algorithm is using the following pseudo code to determine to forward or drop an event

```plaintext
if ((event.TS+ceil(pubInt/2)%subInt [0..i])<pubInt)
    forward event
else
    drop event
```

- The TimeStamp of this specific event
- Publication interval of the variable
- The set of unique subscription intervals for this variable
Temporal Rate Filtering Algorithm (Cont.)

- The function can be broken down to three steps
  - **Shift** all the events timestamps with half the publishing interval
  - The mod of the subscription interval is applied
  - Test if the timestamp is less than the pubInt (the first phase)

Example:
- 4 Publishers:
  - p1, p2, p3, p4
- PubInt: 50ms
- SubInt: 100ms

An Example

Example Topology

Registered publishers and subscribers:
- Pub p0 publishing with interval: 30 ms.
- Sub s0 subscription interval: 90 ms.
- Sub s1 subscription interval: 180 ms. (180/90 = 2)
- Sub s2 subscription interval: 630 ms. (630/90 = 7)
- Sub s3 subscription interval: 240 ms. (240/90 = 2.6667)
- Sub s4 subscription interval: 630 ms. (630/90 = 7)

Talk Outline

- Background and Motivation
- GridStat Framework
- Multicast with Temporal Rate Filtering Mechanism
- Adaptive Mechanism
  - Mode Change Mechanism and Management
    - Hierarchical Mode Change Algorithm
    - Flooding Mode Change Algorithm
- Pattern Mechanism
- RPC Mechanism
- Conclusion
Adaptive Mechanism

- Results published in the following paper:
  

---

Mode Change Mechan. 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

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.

Talk Outline

- Background and Motivation
- GridStat Framework
- Multicast with Temporal Rate Filtering Mechanism
- Adaptive Mechanism
- Pattern Mechanism
  - Introduction to the Mechanism
    - Overview of the Condensation Function
    - Benefits of the Condensation Function
    - Design of the Condensation Function
- 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 its 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:

```
Status_1
...  Condense  ...
Status_N

Publisher
FE
FE
FE
FE
FE
FE
Publisher
Subscriber
use resource
save resources
```

Diagram:

```
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.

Talk Outline

- Background and Motivation
- GridStat Framework
- Multicast with Temporal Rate Filtering Mechanism
- Adaptive Mechanism
- Pattern Mechanism
- RPC Mechanism
  - Ratatoskr RPC Mechanism
    - Architecture
      - Timeliness
      - Redundancy
      - Safety
  - 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

- GridStat Publisher
- GridStat Subscriber
- 2WoPS
- Substation sensors
- Ratatoskr RPC
- Legacy control
- Sensor state monitor
- Protection scheme

Underlying Network:
- QoS Managed Pub-Sub Middleware

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

2 Way over Pub-Sub

RPC Client & Sub Proxy

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
Conclusion

- **GridStat**: status dissemination middleware tailored for critical infrastructures specifically the electric power grid
- **Publish-subscribe** architecture
- Provides mechanisms for
  - Multicast with Deterministic Filtering
  - Adaptive Subscription Sets
  - Pattern Processing Added to Middleware Layer
  - RPC with Timeliness, Redundancy, and Safety
- Network of internal servers managed for QoS
  - Timeliness
  - Redundancy
  - Security

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