Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle *Since 1965*

How to Leverage GOSS: GridOPTICS Software System in a Research Environment

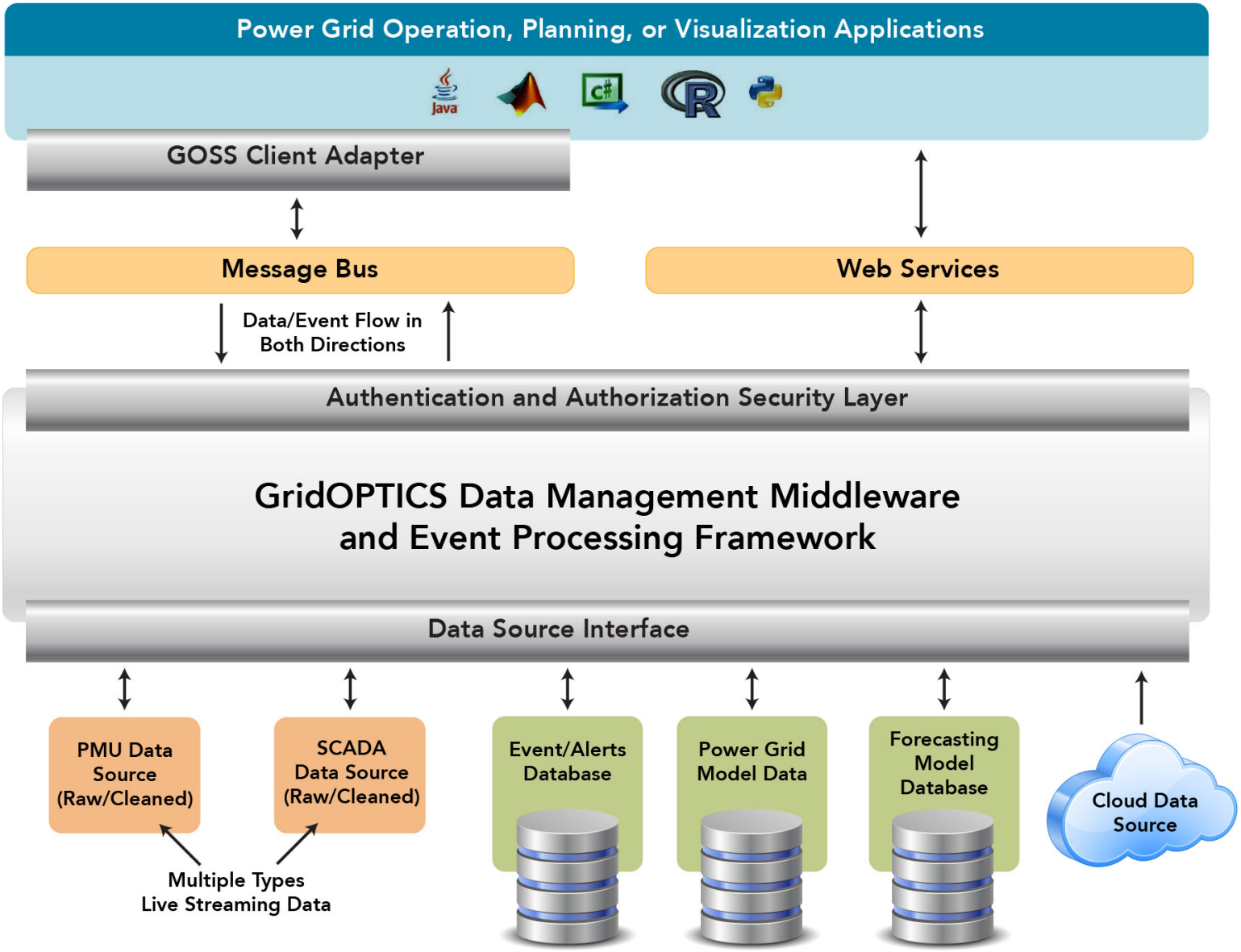
A Novel Software Framework for Integrating Power Grid Data Storage, Management and Analysis

3rd Workshop on Next-Generation Analytics for the Future Power Grid
July 16-18, 2014

Overview

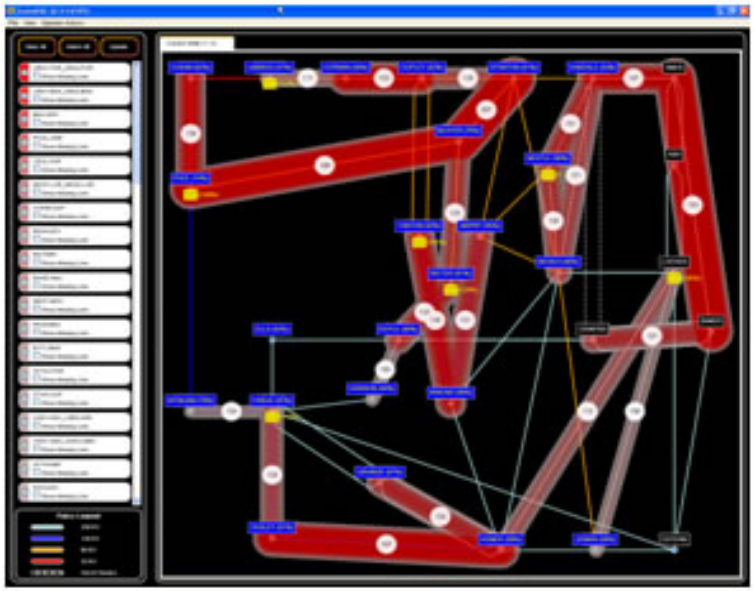
- GOSS is a middleware architecture designed as a prototype future data analytics and integration platform
- https://github.com/GridOPTICS
- What does that mean?
  - Supports heterogeneity – ease of integration with new/existing power grid applications developed in different languages
  - Data source abstraction – separates data sources from applications and provides a unified application programming interface (API) for access
  - Rapid development – Quickly make new data/events available to other applications integrated with GOSS
  - Real-time – subscription to streaming data and events
  - Reliability – provides redundant data access for improved reliability
  - Security – role and data based access control
  - Scalability & Performance

Architecture



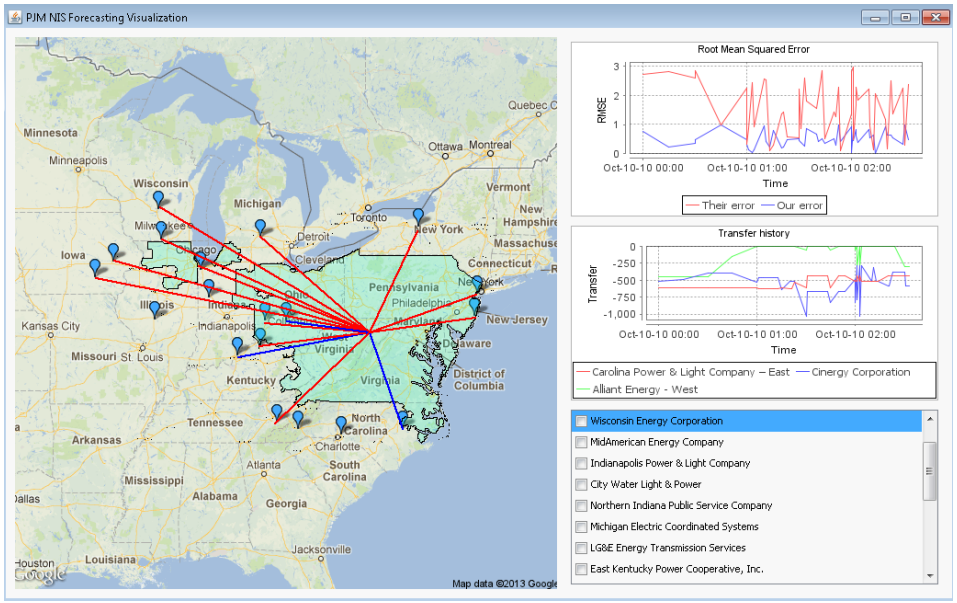
Sample GOSS applications: GCA

- Graphical Contingency Analysis (GCA) - a C# visual analysis application that aids power grid operators and planners to effectively manage potential network failures (N-1)
- GOSS simplified the application by allowing us to combine all input files (power system model, SCADA, power-flow) into a single data source instead of managing multiple files separately
- Data source abstraction allowed GCA to work with time-windowed data
- Application initiates a request for a topology and allows users to select the model to analyze
- Access is restricted by roles. For each utility, access is granted to a set of roles and the user must be in one of these roles in order to access the data for that utility



Sample GOSS applications: NIS

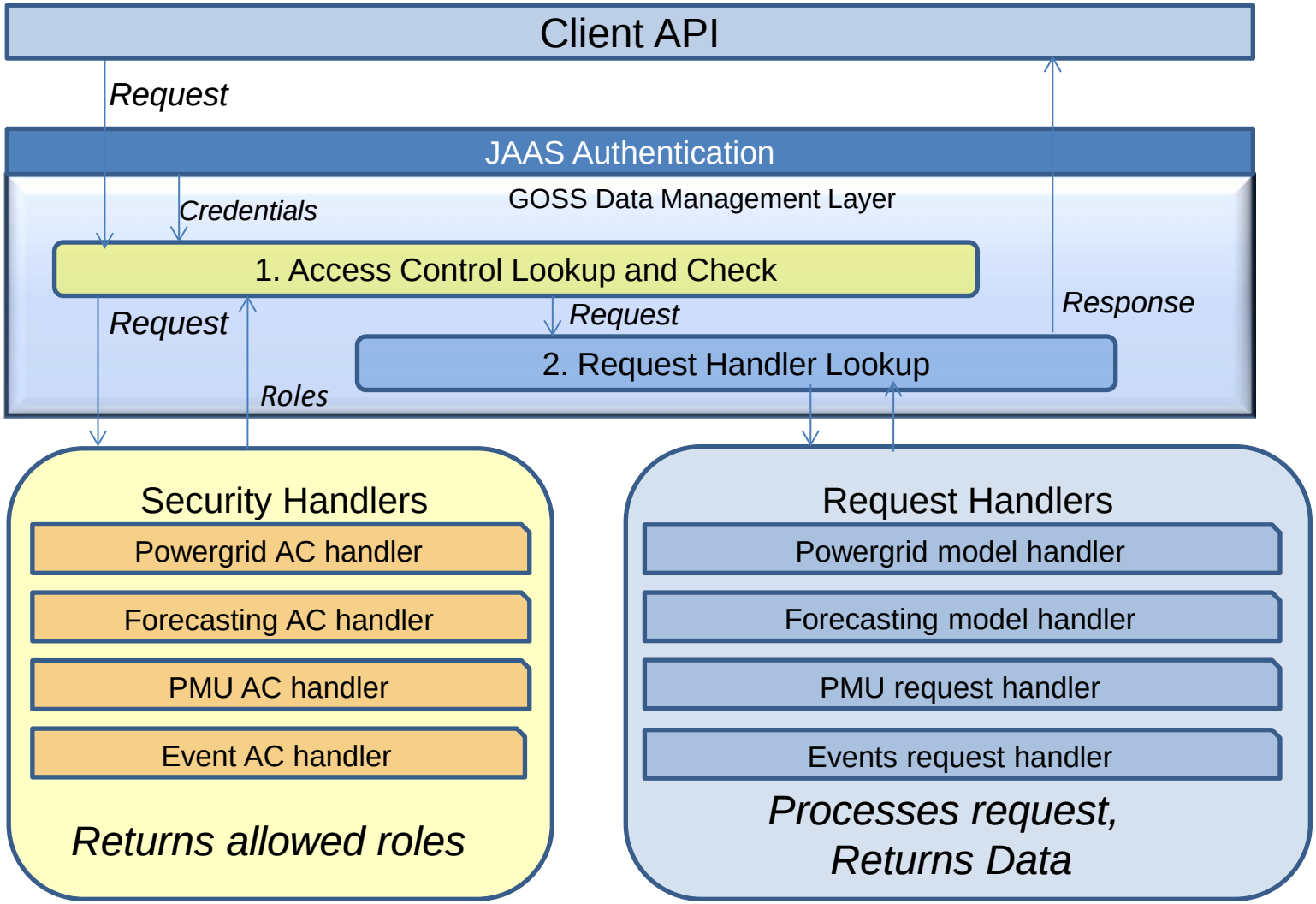
- Net Interchange Schedule (NIS) a MatLab application that displays the sum of the energy import and export transactions between an Independent System Operator (ISO) or a Balancing Authority and neighbors. NIS forecasting (NISF) application was developed to aid the ISOs in economically dispatching the generation resources
- The original application used manually formulated files for the desired time series. With GOSS can use a light-weight client adapter and any time series
- Now able to re-use the algorithm with different data types.
- The input is controlled the same as other PMU data sources, the application will only have access to PMU streams that the user has been granted access to.



Based on Proven Technologies

- Project Development
  - Java
  - Apache ActiveMQ
- Deployment
  - Apache Maven
  - OSGI via Apache Karaf
- Security
  - LDAP
  - JAAS
  - SSL

GOSS Security & Request Flow



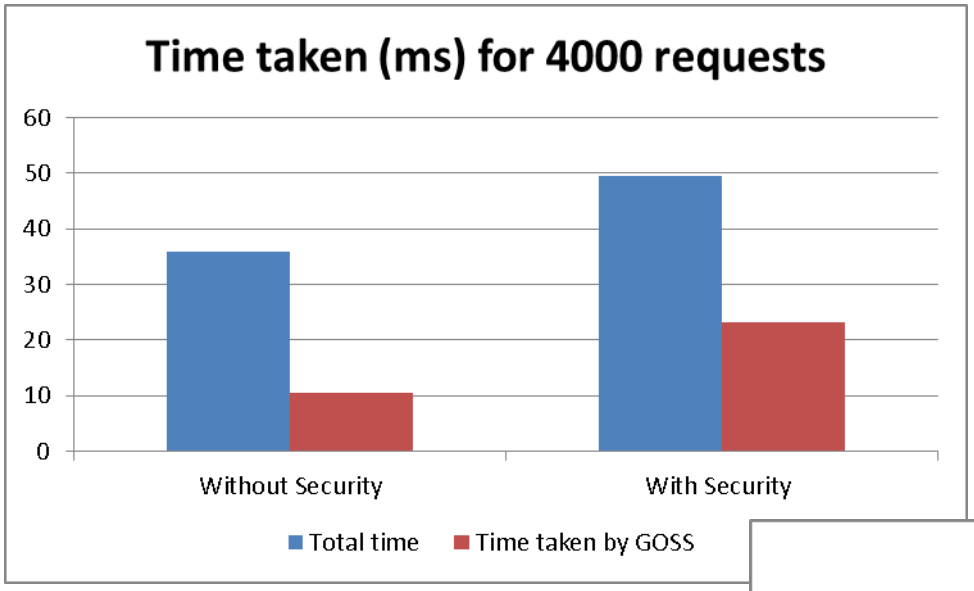
GOSS Authentication

- Authentication – uses widely accepted tools already integrated into communication platform
  - Java Authentication and Authorization Service (JAAS)
    - Easily substitute login modules
  - Lightweight Directory Access Protocol (LDAP)
    - Open, industry standard application protocol for accessing and maintaining distributed directory information services
  - Transport Layer Security/Secure Sockets Layer (SSL)
    - Cryptographic protocols to provide communication security

GOSS Access Control

- Access Control – customizable for each data source
  - Request Specific Security Handlers
- Security Handlers map request to list of allowed roles
- User verified for correct role access
- Multi-role Access
  - Request combining multiple sources
- Handler implementations for common data types
  - Time series data

Initial Performance Benchmarking

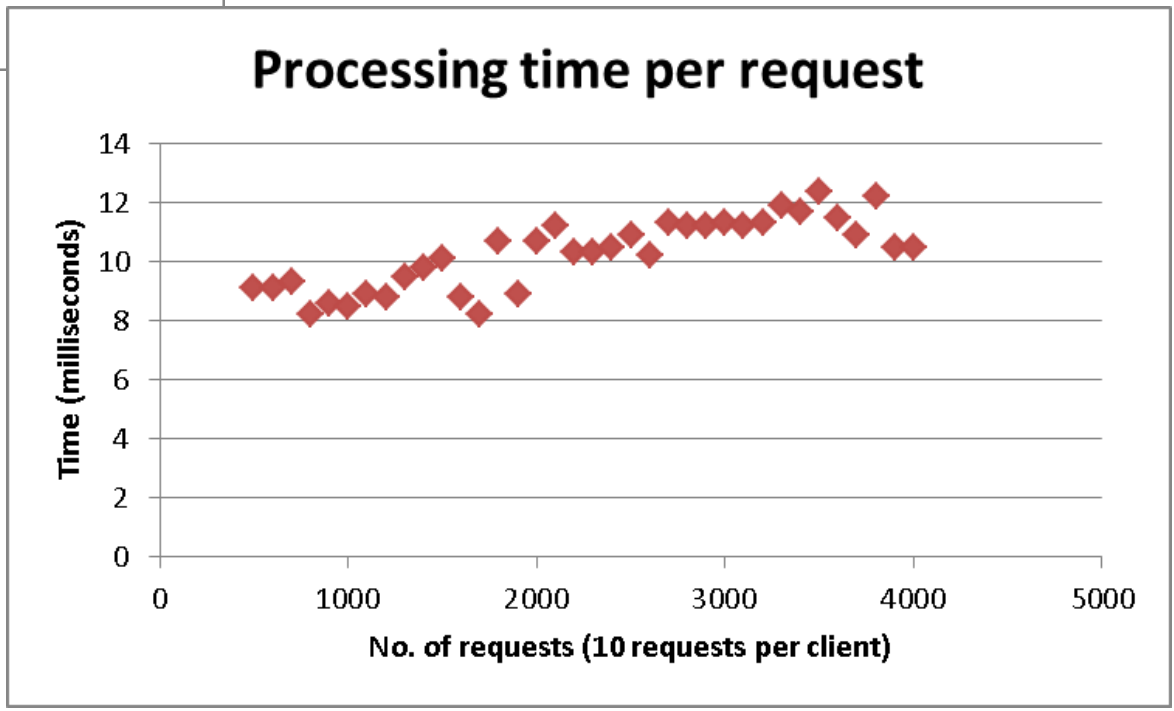


Test 1: Comparison of *average* time taken by data store and GOSS individually in total READ request processing time

- Data size ~700 KB
- Number of requests = 4,000
- Number of Clients = 1
- Each client executed in separate thread.

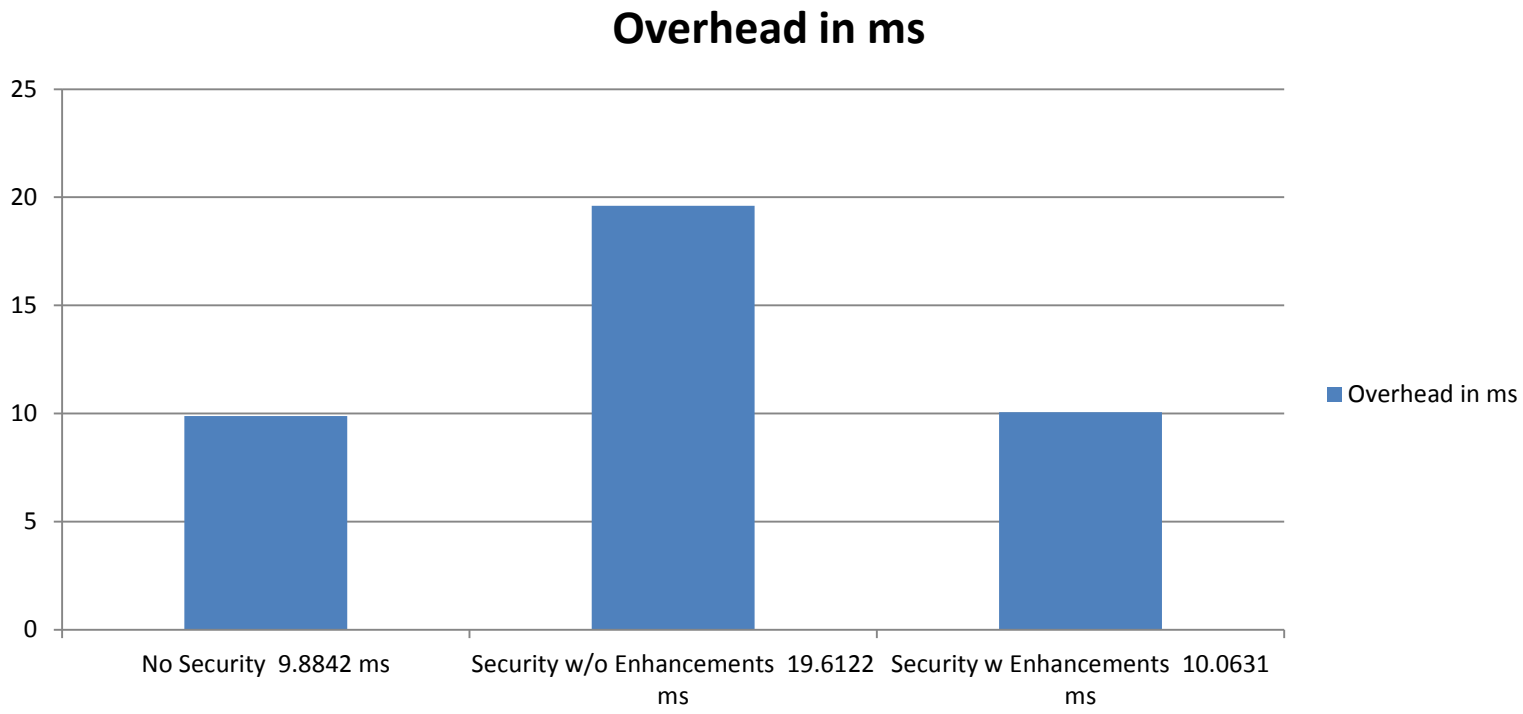
Test 2: Request processing time with increasing number of concurrent READ data requests

- Each client sends 10 requests
- Data size ~700 KB
- Each client executed in separate thread



Synchronous Performance After Enhancements

- GOSS Overheads using same method as previous slide
  - Before enhancements, security adds almost 100% increase
  - After enhancements, reduced to only ~10%



Performance Benchmarking Analysis

- Per Client Request, processing time is stable even with increasing number of clients
- Scales well with increasing load
- Total time spent inside GOSS includes not only data access but also:
  - Data routing between data source and application
  - Query conversion. Generic query format to data store specific query (e.g., SQL)
  - Result conversion. Converting the results to format requested by the application (including object transformation). Eg., JSON, XML, Serialized Object, etc.
  - Security and access control
- Tests show results in “synchronous” access mode. Asynchronous access hides most of these latencies via pipelining.
- Real-time applications likely to use either event-based or asynchronous access.

Future Tasks

- Synthetic Data Generation
  - Modify the code as needed to perform research
  - Ability to interface with other applications with lower cost
  - Simulators will be tied to GOSS
- Fine Grained Security
  - Certificate based authentication
  - Improved multi-domain support
- HPC Integration
  - Access data and launch simulation
- Fault tolerance

GOSS Team

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

- Bora Akyol bora@pnnl.gov
- Poorva Sharma poorva.sharma@pnnl.gov
- Craig Allwardt craig.allwardt@pnnl.gov
- Mark Rice mark.rice@pnnl.gov
- Tara Gibson tara@pnnl.gov

Questions?