RASSMon: Realtime Asynchronous Service Status Monitoring

Bianca Neagu, **Corina Dulea** and **Horia V. Corcalciuc**
Department of Computational Physics and Information Technologies
Horia Hulubei National Institute for R&D in Physics and Nuclear Engineering (IFIN-HH)

September 17, 2017

Monitoring Systems

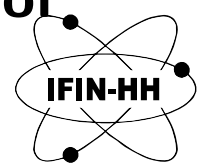
Systems monitoring can be performed in two ways:

- **monitoring agents that run locally to a site** (implicitly, with full access)
- **by launching jobs remotely and correlating the response with known results** (partial access)

Monitoring agents that run locally *does not pose a problem* because:

- full access to the system allows changing parameters conveniently
- concensus amongst results can easily be attained due to faster feedback

the creation of a local monitoring system **becomes an exercise of software design where both the monitored systems and the monitoring software can be altered** to attain desired results.



Remote Monitoring

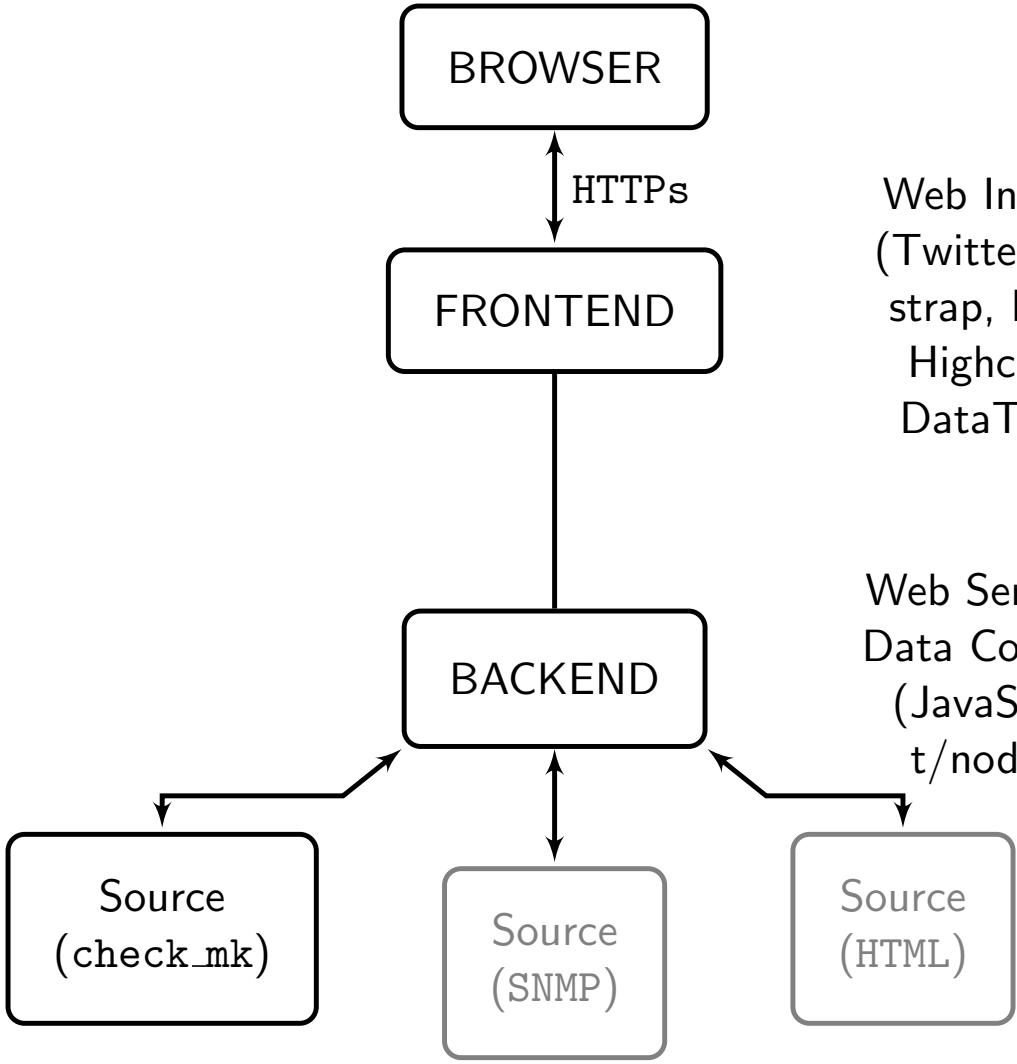
What if only **partial access** is granted by a remote system that must be monitored?

- More than often there is no way of acquiring data in a standardized format (ie: display-only web-pages). For instance, locked-down CERN ETF "Check MK" instance.
- Existing monitoring systems **rely on an agent** that must be run remotely to provide the desired data.
- Monitoring systems ("monit", "NaGIOS", "SeaLion", etc.) are designed to extract system-centric typical data (CPU, Memory, Networking, etc.) rather than **project-oriented data**.

Whilst agent-based monitoring is conveniently implemented through a **producer-consumer** model, monitoring sites with no access can only be performed through some form of **polling**.

IFIN-HH

RASSMon

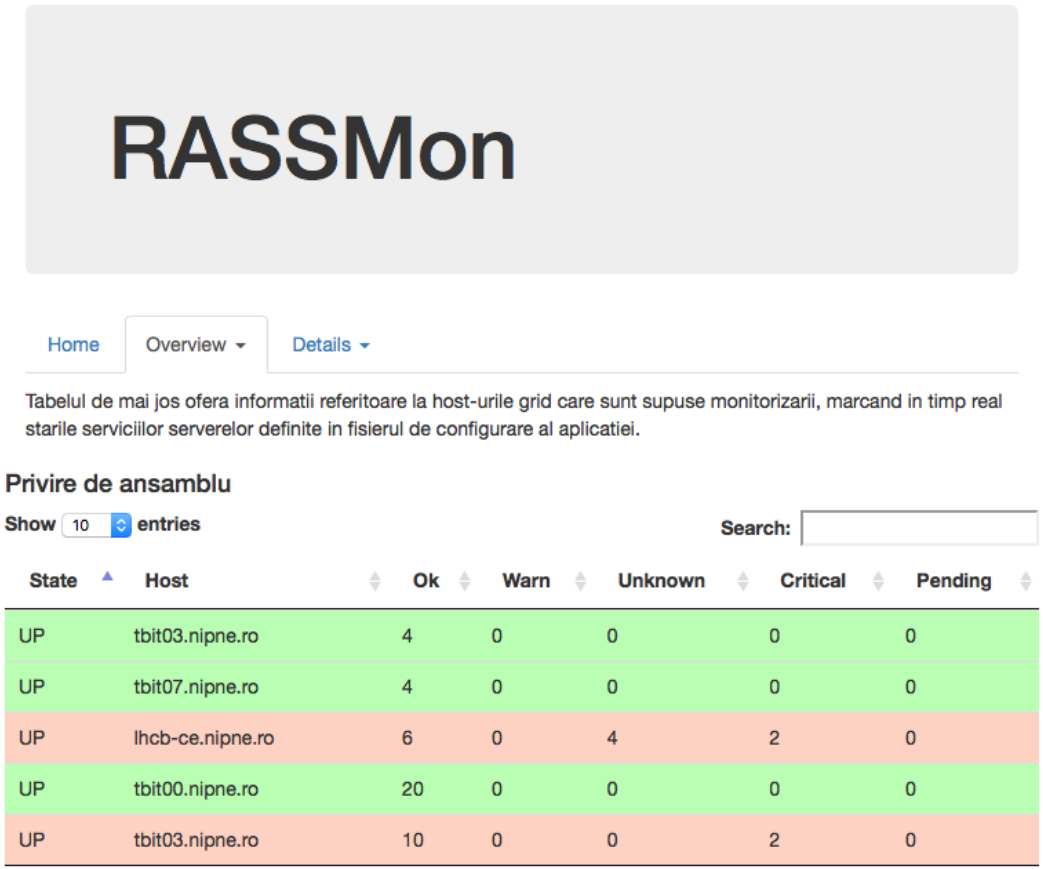


Web Interface (Twitter Bootstrap, Bower, Highcharts, DataTables)

Web Server & Data Collector (JavaScript/nodejs)

IFIN-HH

Website Frontend - Server Status Overview

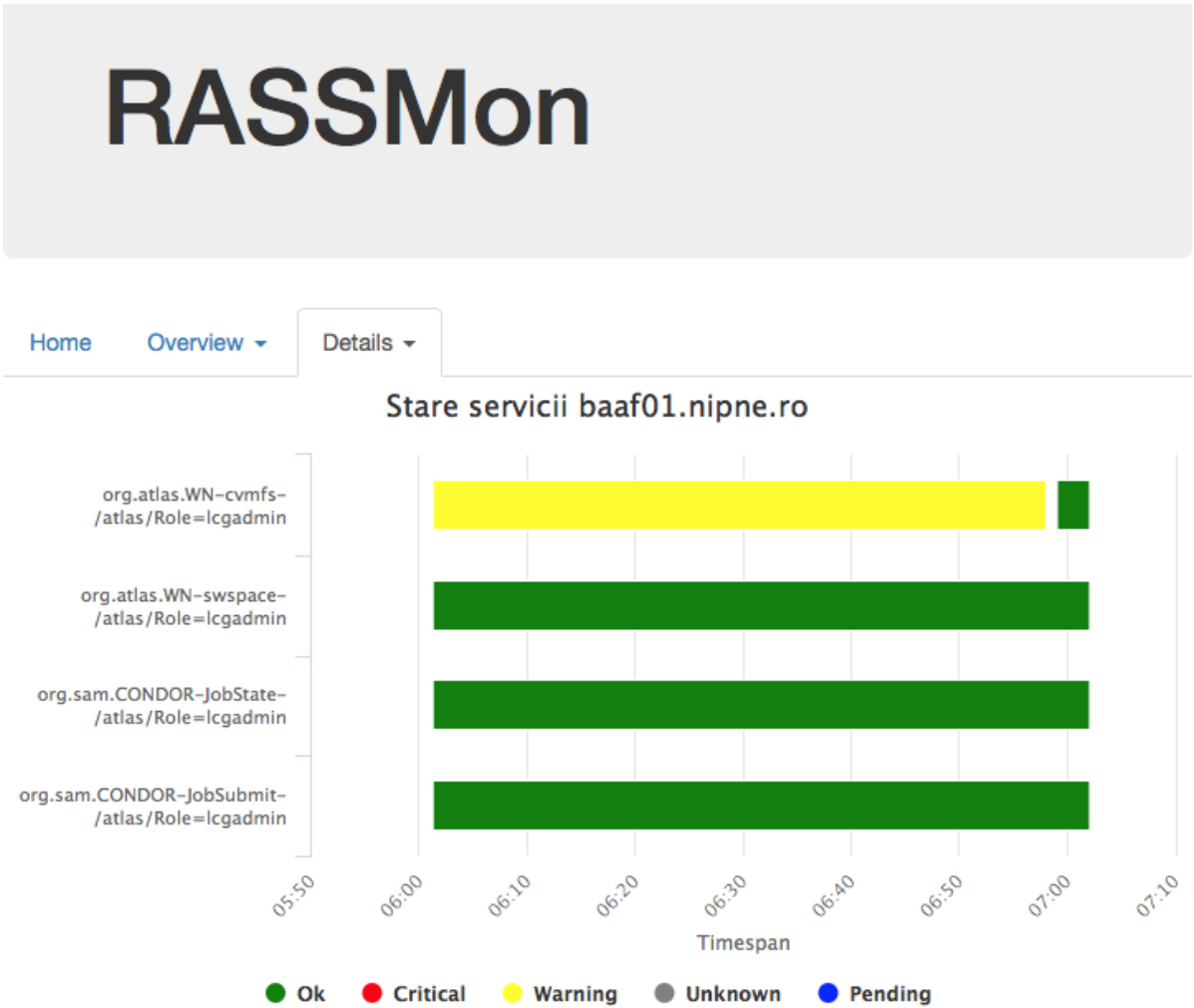


On the frontend:

- **RASSMon** provides a table overview of all servers with the color matching the status (green for all services running, red if at least one service is down).
- Each server can then be accessed in order to display a stacked-bar chart overview of all the services running on that server along with their corresponding statuses.

IFIN-HH

Website Frontend - Status of Services



- All data is retrieved and the display is updated in real time allowing operators to see and react to any changes in server availability.

IFIN-HH

The Charms of Asynchronicity

Definition

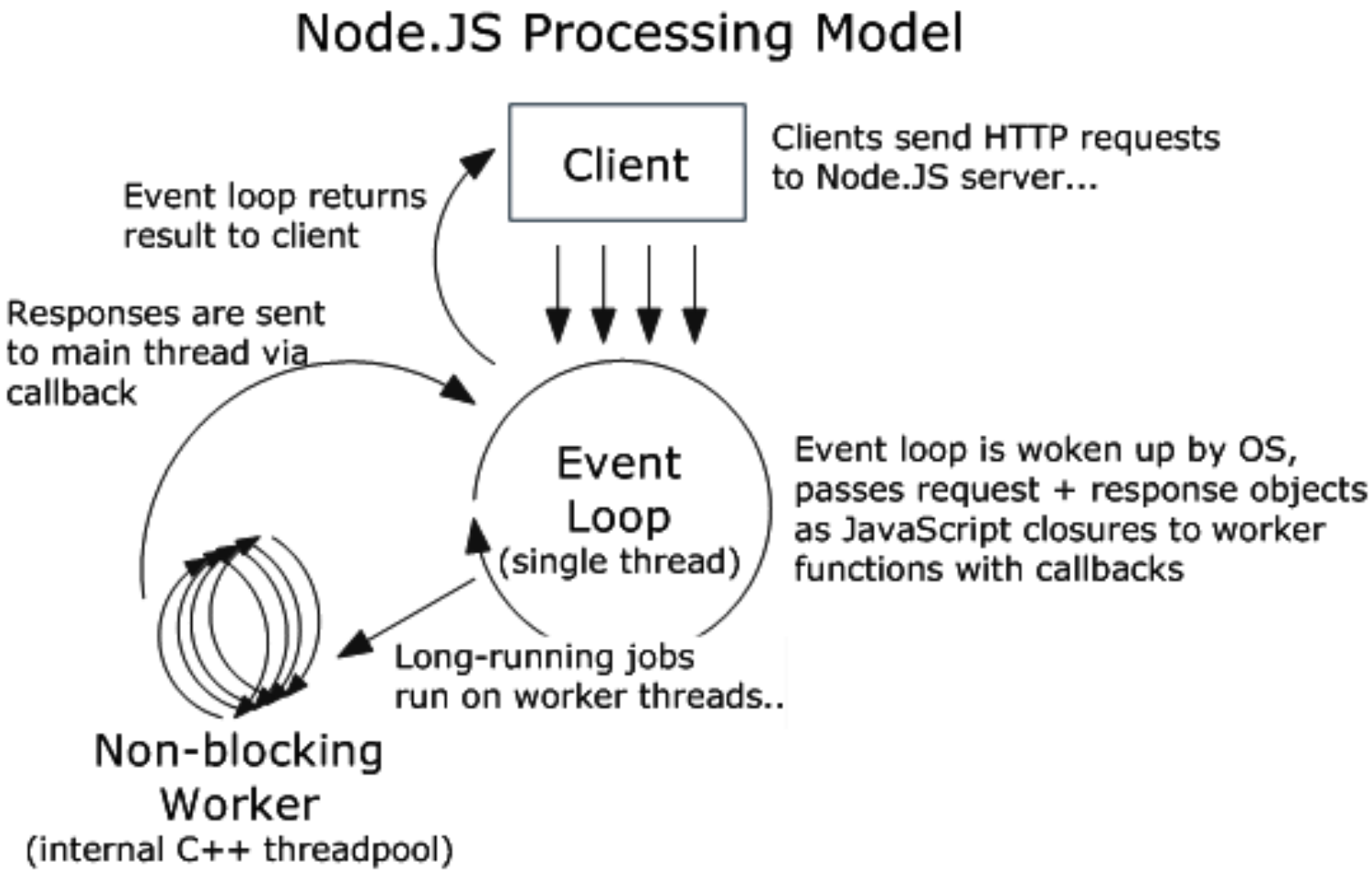
The difference between asynchronicity and a thread-based approach is that threads allow execution of code **in parallel** whilst asynchronicity means that code is executed **at a later time when the scheduler deems it convenient** and **results are waited upon without blocking the main program flow** thereby **eliminating well-known long delays**.

- This differentiation is **semantic** and appears transparent to a programer that, in both cases, will place work to be executed on an event loop and will retrieve results at a later time via some form of synchronization (**promises** for asynchronicity)
- Known instances of long delays that do not necessarily warrant the use of threads are cases where data has to be retrieved remotely via the network subsystem (**polling**) - for instance via HTTP AJAX requests.

RASSMon is built using "Node.js" (**single-threaded event loop**) and manages to pull data from many sources, serve pages interactively to connecting web-browsers as well as keep track of multiple non-local sites.

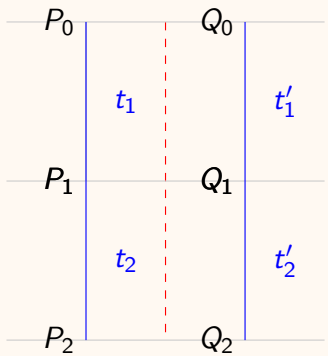
IFIN-HH

Processing Model

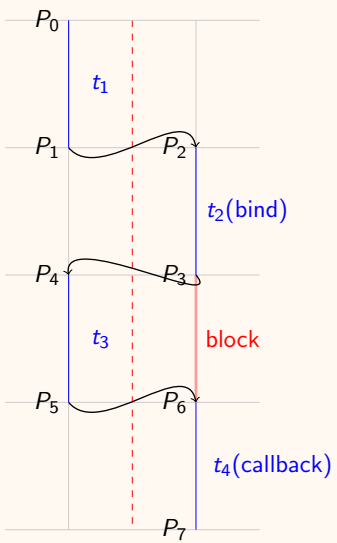


Reasoning about Asynchronicity

Parallel (concurrent) Control Flow



Asynchronous Control Flow



Using traces (de Roever et al.) control flow can be diagrammatically explained as a sequence of composable traces $t = t_1 \cdot t_2$, $t = t'_1 \cdot t'_2$ that describe state transitions within a program where the states satisfy intermediary pre-and postconditions ($P_{0,1,2,3}$ and $Q_{0,1,2}$).

Parallel execution allows for transitions (**in this case, without interleaving**) between states in parallel, described by two separate sets of traces $t_{1,2}$ and $t'_{1,2}$ whereas asynchronous control flow will bind to events on the main execution thread and then process the continuation with a callback on the same thread.

IFIN-HH

Data Sources

RASSMon was designed to be used for retrieving data from sites that do not offer a programatic way to retrieve information - for instance, display-only websites without export features.

- At the current development stage, given the requirements, **RASSMon** does not attempt to replace other monitoring solutions but rather offer a complementary solution in cases where no data export is available.
- Currently **RASSMon** implements data retrieveal from a "Check MK" instance via a "Node.JS" plugin. Plugins *have to be written depending on the data format* that a site offers for exports - for instance "Nagios" offers JSON as well as CSV (RFC4180 compliant), such that the former may be preferred due to the interoperability with JavaScript.

IFIN-HH

Relational Databases vs. Flat File Storage

RASSMon remotely monitors the CERN ETF "Check MK" instance, retrieves a CSV-formatted snapshot of the status of services running on tracked servers and stores the results as JSON serialized data to the filesystem.

RASSMon does not use popular relational databases such as "MySQL", "SQLite" or "PostgreSQL" because:

- The main advantage of databases is to use the relational features (built-in data processing functions, structured query language) that are not needed for **RASSMon** because all the processing is offloaded in real-time to the client browser. **Databases should not replace storage**.
- Service snapshots that are retrieved from CERN have a relatively short lifespan without the need for archival. The monitored timespan **can range from a few minutes to, at best, several days** and **old snapshots are periodically discarded**.
- Snapshots are serialized to the filesystem in an incremental fashion such that no **atomicity** must be guaranteed for either reading or writing to separate files (**eliminating the need for locks** can greatly increase performance).

Serialization and Deserialization

One of the project goals of **RASSMon** has been to only use well-defined data formats. "Check MK" provides CSV exports but the format of the data is not compliant with any standard namely the one specified in "Common Format and MIME Type for Comma-Separated Values (CSV) Files" (RFC4180).

Pulling data from the CERN ETF website with **RASSMon** will take place in the following fashion:

1. **RASSMon** connects to the CERN ETF "Check MK" instance.
2. For all servers that have to be monitored, the data is exported via the "Check MK" CSV export feature.
3. The CSV data is then processed **leniently** and serialized to JSON and stored on the filesystem.

The decision to use JSON for serialization is due to **RASSMon** being written in JavaScript (with CSS and HTML for markup) such that the interaction with JSON is seamless compared to other structured languages (ie: XML).

Configuring RASSMon

RASSMon YAML Configuration Excerpt

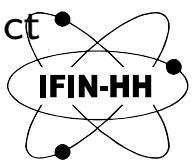
```
# A list of servers to check for the check_mk component.
check:
  - arc.nipne.ro
  - baaf01.nipne.ro
  - baaf02.nipne.ro

# Whether to use authentication for the frontend.
authentication: false

# If authentication is enabled, the path to a htdigest file.
credentialsPath: auth/passwd
```

- **RASSMon** reads the configuration file once at startup and proceeds to fetch data and serve webpages once clients make requests.
- The configuration file is based on **YAML** instead of JSON:
  - YAML can contain comments that are convenient to describe configuration parameters.
  - YAML is a humanely readable language without too much syntactic sugar.
- By adding or removing servers from the list, **RASSMon** dynamically adapts and renders the tables and charts.
- Both the frontend and the backend read the configuration file.

IFIN-HH

Candidate Points and Mitigation

- JavaScript does not allow direct memory access which removes the possibility of a **buffer overflow attack** with follow-up **code injections**. At best, an exception can be raised that would be handled in the code.
- The **RASSMon** frontend relies on the client browser to process the information such that any attempt to circumvent security through user-input (ie: **ReDOS**) would just lead to a slower browser.
- When files are served by the backend, all requests are jailed to the frontend website directory and requests for any other files are rejected thereby eliminating the possibility of a **path traversal attack** - **RASSMon** splits the server / executable directory from the web frontend directory.
- At the time of writing, the **RASSMon** frontend only serves static pages and the backend implements only a subset of the HTTP protocol (mostly `GET` requests and caching) without giving the ability to clients to insert any data.
- The frontend webserver uses **HTTPs for all requests** with custom certificates that can be generated based on a certificate template to encrypt all communication between a browser and **RASSMon**. Optionally, **RASSMon** supports authentication via Digest over HTTPs allowing deployments to restrict access.



Questions

IFIN-HH