**SEVENTH FRAMEWORK PROGRAMME**

**THEME ICT-2013.3.4**

Advanced Computing, Embedded and Control Systems



Execution Models for Energy-Efficient

Computing Systems (EXCESS)

611183

**D3.3**

**First prototype of monitoring framework for the**

**Conventional HPC and Movidius platforms**

Yosandra Sandoval

Konstantinos Dolkas

FP7-coo-CMYK

Date of preparation (latest version): dd.mm.yyyy

Copyright © YYYY The EXCESS Consortium

The opinions of the authors expressed in this document do not necessarily reflect the official opinion of EXCESS partners or of the European Commission!

# Document Information

|  |  |
| --- | --- |
| **Deliverable Number** | D3.3 |
|  |  |
| **Deliverable Name** | **First prototype of monitoring framework for the conventional HPC and Movidius platforms** |
|  |  |
| **Authors** |  |
|  |  |
| **Responsible Author** | First\_Name Last\_Name (ORG)  e-mail:  phone: |
|  |  |
| **Keywords** |  |
|  |  |
| **WP/Task** | WP3 / Task 3.3 |
|  |  |
| **Nature** | R = Report, P = Prototype, D = Demonstrator, O = Other |
|  |  |
| **Dissemination Level** | PU = Public  PP = Restricted to other programme participants (including the Commission Services).  RE = Restricted to a group specified by the consortium (including the Commission Services).  CO = Confidential, only for members of the consortium (including the Commission Services). |
|  |  |
| **Planned Date** | dd.mm.yyyy |
|  |  |
| **Final Version Date** | dd.mm.yyyy |
|  |  |
| **Reviewed by** | Person1 (ORG1), Person2 (ORG2), .. |
|  |  |
| **MGT Board Approval** | YES | NO |

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Person** | **Date** | **Comment** | **Version** |
|  |  |  |  |
|  | 25.04.2014 | Fist input | 0.1 |
|  |  |  |  |
|  | dd.mm.yyyy |  |  |
|  |  |  |  |
|  | dd.mm.yyyy |  |  |
|  |  |  |  |
|  | dd.mm.yyyy |  | Final |

# Abstract

Please provide a brief summary of deliverable, a brief statement of main results/points of interest, nature of deliverable (summary of technical papers, software overview, etc.), reference to software, papers and/or specifications found outside of deliverable.

For update deliverables: summarize main changes over last previous version.

# Table of Contents

1. Document Information 2

2. Document History 3

3. Abstract 4

4. Table of Contents 4

5. Executive Summary 6

5.1. Main results and achievements 6

5.2. Work package and tasks, relations to other work packages 6

5.3. Progress and updates on related deliverables 6

5.4. Open issues and future work 6

6. Introduction 7

7. Requirements 7

8. State of the art 12

8.1. Application performance management 13

8.2. Measuring application performance 13

8.3. Current Issues 13

8.4. The APM Conceptual Framework 14

End User Experience – (Primary) 14

Runtime Application Architecture (Secondary) 15

Business Transaction (Primary) 15

Deep Dive Component Monitoring (Secondary) 16

Analytics/Reporting (Primary) 16

8.5. Nagios 16

8.6. Kieker 16

8.7. Zabbix 16

8.8. OpenNMS 16

8.9. Ganglia 16

9. High level architecture 17

10. Implementation methodology 17

10.1. Agile principles 17

10.2. Agile Overview 18

11. Technologies selected 19

11.1. Client Side 19

C library 19

libcurl 19

11.2. Server Side 19

Node.js 19

ElasticSearch 19

12. Initial implementation results 19

Walkthrough the screens(UI) 19

Backend details 19

13. Possible ways of extending 19

14. References 19

15. Glossary 19

# Executive Summary

Longer deliverables may require a 1-2 page executive summary; possible structure could be like this:

## Main results and achievements

…

## Work package and tasks, relations to other work packages

…

## Progress and updates on related deliverables

…

## Open issues and future work

…

# Introduction

The monitoring framework (MF) has been defined and implemented in order to support energy analysis at runtime. It collects energy- and performance-related data from various sources. That data is collected, formatted and categorized, and finally used for visualization, analysis and prediction.

A set of energy-aware HPC benchmarks is developed in the same work package (WP3) in order to provide feedback to the development of the monitoring framework. During the benchmark execution the monitoring framework collects the information about the electrical power consumption of different hardware components (CPU, GPU, memory and so on) as well as the whole computational node for the different computations kernels. Based on these data and the achieved performance, a power consumption-predicting model was developed for different hardware components. This system is also used for checking the system hardware built-in “power meter” for their accuracy (for example Intel PCM registers). The prediction model of power consumption will give the application developers the option of simple calculation of the power consumption with enough precision and without complicated measuring equipment since the hardware built-in “power meter” accuracy will be known.

# Requirements

**Previous work**

Add a small paragraph of the work that was done in eco2clouds

Cross WP discussions took place in order to define the requirements for the monitoring framework. WP1 and WP2 will be the main users of the system and the main purpose of these discussions was to identify the key issues, that needed to be addressed and functionality that the systems should include.

The requirements have been categorized as follows

Implementation detail

* The MF should be a C library

Functional Specifications

* Raw data should be gathered
  + Energy (
  + Performance (
  + Time ( Start and End )
* Raw data should be possible to be requested by 3rd party systems (WP predictors)
* The gatherers frequency should be configurable (starting with 1/sec)
* Historical and current raw data should be available

Non-functional specifications

* MF should be closely coupled with the runtime system,
* Raw data should be available real-time or close to real time for on-line training,
* MF should be as less intrusive as possible,

**HLRS:** It will be based on plug-ins-> extendible

- It should be portable, i.e. should be deployable at other sites and hardcode as little as possible (make all assumptions explicit and configurable). It should work properly on systems that have no additional measurement hardware available (and use only counters in such cases).

**HLRS:** There are some restrictions for the portability e.g. Intel counters

UiT: What about the Movidius platform?

HLRS: There should be no problem to deploy the monitoring framework on it. But we have to investigate about this a little more.

- It should be retargetable, and ideally be generated for each EXCESS target system from its formal description in XPDL (XPDL will be defined later in WP1). Example: If my machine has no GPU, it should still be possible to use the Monitoring Framework without rewriting it.

- It should allow to write time- and energy-related data traces to files for later processing and visualization.

- It should come with a default visualization tool, which allows to visualize energy properties e.g. by color coding of power consumption in task boxes in a processor-time diagram.

- It should allow to collect the data of remotely executed code remotely via an internet connection (web service? ftp?) (this is probably of secondary importance only, as one could work online via ssh instead)

- It should allow to export data in a format that can be processed by other data analysis tools, such as octave etc. A simple format such as csv (comma separated values, used by many tools) might be considered as one option.

- It should allow easy integration with other tools such as PAPI etc.

- It should support simultaneous sampling from multiple different hardware measuring devices and software methods, and correlate time stamps for these devices.

HLRS: Can you explain this a little better?

LIU: that you should be able to report the various metrics of the system at the same point in time, in order to have a snapshot of the system. This could be a problem with the monitoring framework taking measurements every 10 sec.

HLRS: to be able to calculate the energy consumption trough the counters and in the same time gather the information from the actual hardware energy consumption and correlate both... That’s one of the points in the diagram circulated today, where it reads: “Automatically triggering of remote hardware energy measurements”. So we can be sure that the metrics are taken exactly at the same time

- If possible, add an API to notify the hardware energy measurement devices to start and end recording.

- Besides counters, high level metrics such as throughput should be monitored, for GPU these numbers can be directly extracted by Nvidia CUPTI library or Nvidia Visual profiler.

LIU: The throughput may be derived from other metrics and it not so critical to store it because it can be calculated

HLRS: For now we decide to exclude the throughput. It may be discussed later on.

- It should be configurable by what to monitor, and extensible.

- It should be possible to filter trace data based on some property, e.g. a time interval of interest.

- It should allow to run the same application multiple times in sequence and discard the first K runs, K being a parameter, in order to eliminate warm-up effects.

- It should allow to correlate energy and time measurements.

- It should allow to relate measurement values to tasks, not to whole threads or processes. This needs to be coordinated with the EXCESS runtime system (i.e. task API, dynamic scheduler, data management etc., see StarPU). LIU can include this functionality also in the generated task API code in WP1 composition tools and SkePU backends by instrumenting the generated code with calls to the monitoring API.

HLRS: It is not possible right now for the energy part and the only possible way is that we provide pointers in time indicating when a task starts and ends, and provide this information to the user to analyze.

LIU:

- It should allow to measure energy for different hardware components such as the following (please feel free to add more):

+ CPU per socket

+ CPU-specific counter values. (flops, branches, instructions executed, memory load and store accesses, L1, L2, L3 misses and hits)

+ GPU for each GPU in the system (flops, ...)

+ QPI quick path interconnect

+ Memory

+ Hard disks

+ Network interface

+ Fans

+ Temperature

+ Execution time

**HLRS:**

Regarding energy metrics: We can measure RAM and CPU power consumption together (per socket) given the hardware restrictions of the cluster. In case this restriction does not exist (the hardware to gather the respective energy metric is available) the Monitoring framework will be able to gather both metrics separately and send them to the server. The API will provide only the specific metrics that should be gathered. It is possible to takes measures from the GPU. We will investigate how the hard disk measuring can be done. The network interface and Fans are not totally clear. One metric that was not mentioned in the list above is the total energy consumption of the node and we will consider it on the Monitoring Framework.

Regarding the performance metrics: we will get the performance counters (CPU usage, memory percentage and GPU).

For the other metrics: there were no more metrics mentioned on the telco, only Temperature listed above.

This list of relevant features will be one outcome of D1.1 investigations currently done at LIU, CHALMERS and UIT, so we will know more in 3-4 weeks.

**HLRS:** What is the expected minimum time interval to gather metrics (time resolution of the metrics) for the Monitoring Framework?

We proposed 10 sec for performance metrics time resolution, and may make this interval smaller as the project advances

LIU: This time interval has to be configurable (software defined). The metrics have to be related with individual task invocations.

HLRS: It will be configurable.

LIU: metrics time resolution in millisecond should be possible.

HLRS: Millisecond resolution would be too intrusive and consume too many resources.

HLRS: we propose that the minimum metric time resolution should not be smaller than 1 sec. This already produces around 1-3% overhead.

UiT: What about the minimum interval for energy measurement?

HLRS: For the energy metrics we can have time resolution of less than 100 milliseconds. This should be possible because the measurement system is not intrusive. It is an external node which takes the measurements.

LIU: We should be able to align energy and performance metrics (so if energy metrics have a resolution of milliseconds, then the performance should also have this resolution or similar)

HLRS: for the first version of the Monitoring framework we can only promise: we will begin with a big interval (10 sec)

UiT: The time interval has to be configurable depending on the metric type

HLRS: Ok, but the interval will not be in milliseconds for performance metrics.

**Regarding the other questions:**

> Task 1.2 mentions "This includes the design for compositionality of energy models of software building blocks”. What will be needed from the monitoring framework for this task?

see above

**HLRS:** - Where will those tests be performed (LIU Server, HLRS cluster, …)?

**LIU:** on every EXCESS server (HLRS, LIU, CHALMERS...) and hopefully even on Movidius based systems if applicable, within the project. The long-term vision is of course that the system will be so portable that the final (open-source) EXCESS software framework can be installed on a large spectrum of Linux-based third-party machines, too.

**HLRS:** - Do we “HLRS” need to be monitoring for each individual building block?

**LIU:** See above - LIU can fix it for tasks by instrumenting our generated code appropriately, but you should also consider monitoring for individual tasks, not only for threads and processes, in order to make your framework more general and versatile.

**HLRS:** - What kind of information do we have to collect for each building block?

**LIU:** See above

**HLRS:** - What are those building blocks?

Software building blocks are tasks, i.e. calls to software ("PEPPHER") components or instantiated SkePU skeletons. For further information, please read our recently submitted survey paper in the SVN at WP1-programming-models/papers/kessler-racing14-v1.pdf

> Task 1.2 mentions "The entire modelling framework should be general enough so that it can be instantiated for the case studies and systems used for evaluation by the partners in the project.”

**LIU:** see above for further elaboration

**HLRS:** - Do we have a draft on the "modeling framework" specification?

No, this will start with the D1.1 and in 4 weeks we will know more.

**HLRS:** - What are the proposed structure/specs of it?

**LIU:** It does not exist yet because it is a future deliverable (D1.2, PM12). At the moment we have to prioritize D1.1.

**HLRS:** - Should the monitoring framework follow the same principals?

**LIU:** Yes, if possible.

> Task 1.3 mentions "relevant metrics such as average execution time, energy, and quality/accuracy properties of components”

**HLRS:** - Can you please define average execution time? Is that during execution? Is it for the total duration or just a minute average?

**LIU:** It is the expected (predicted) value for elapsed execution time for a task's execution.

**HLRS:** - Can you please define energy? What is the metric used? Is it versus performance? Is it CPU, Memory or overall?

**LIU:** It is expected energy consumption for a task. The overall objective function will be based on overall energy, but we will probably have to break it down by different HW components and energy-affecting factors, see forthcoming D1.1.

**HLRS:** - And finally what about quality and accuracy? How should those be monitored/measured?

Quality/accuracy (e.g. result accuracy, degree of data sortedness etc.) was taken up in the proposal mostly for tactical reasons and will probably not be considered as an important metric in WP1. In cases where it is relevant, it should be added as a user-defined metric for individual (SW) components; for simplicity, constraints on quality/accuracy could be handled without extensions of the component model syntax by Conditional Composition (see our recent MULTIPROG'14 paper, presentation slides now in the SVN at WP6-dissemination/presentations/DastgeerKessler-MULTIPROG14.pptx

> Task 1.4 mentions “This will allow to balance trade-offs between energy and time.”

**HLRS:** - Again as above what should the monitoring framework provide for task 1.4?

**LIU:** No special requirements for Task 1.4, it will use the energy model of Task 1.2 for global optimizations.

**HLRS:** - What about performance?

**LIU:** Same. Energy and performance need to be considered together, usually one is a constraint and we optimize for the other.

> Task 2.1 mentions “Investigate and model the trade-off between energy and performance”

**HLRS:** - What is the monitoring framework expected to provide?

**UiT:** - The monitoring framework should provide a common API (in C/C++) for threads to collect at runtime energy consumption, in addition to conventional performance metrics, of key hardware components in EXCESS platforms such as CPU, GPU, Movidius' SHAVE, memory system (e.g. DDR, L2/L3 cache, scratchpad CMX), interconnect.

**HLRS:** Energy per threads is not possible. The measure of the energy consumption will be taken on CPU and memory together. We can not discriminate what energy is consumed by each component.

LIU: One possibility is to extrapolate the consumption, to assign a percentage to CPU and memory from the total energy consumption (proportional prediction)

- The API should allow threads to select different sampling steps (i.e. clock cycle interval). The framework should be able to keep sampling steps as short as possible in order to detect most power variations/phases in EXCESS platforms.

**HLRS:** Minimum 1 sec.

- The framework should not interfere with threads execution. Namely, the energy consumption and performance of the execution should be the same regardless of whether the monitoring framework is turned on or not.

> Task 2.2 mentions “This task will constitute the interfaces and libraries for inter-process communication and data sharing on EXCESS new platforms.”

**HLRS:** - Should the prediction model be part of these libraries? Is there a draft/initial idea of this?

**UiT:** The prediction model as described in Task 3.1 is a component of the monitoring framework to improve its precision and speed, and therefore the model will not be a part of the libraries for inter-process communication and data sharing. These libraries will interact with the monitoring framework and its prediction model only through the common API mentioned above.

**\*General\***

**HLRS:** - Should the monitoring framework be responsible for actions/decision? or just for monitoring?

**LIU:** Only for monitoring. It should never take any actions by itself because that would interfere with the global optimizations that we will develop for EXCESS in WP1. Of course, if you want to reuse your framework outside EXCESS you might add optional control mechanisms. The runtime system might take some actions (as configured by the upper software layers), but that is outside the monitoring framework.

**HLRS:** - Who will be acting based on the results of the observation e.g. tradeoff during execution?

**LIU:** The generated composition code executing as part of the generated application code. And maybe the runtime system, see above.

**HLRS:** - When prediction model expects a “bad" result, what should happen?

**LIU:** What do you mean by "bad result"?

If the prediction is of low accuracy, the worst thing that can happen is that a suboptimal implementation variant or resource allocation is selected. The application will execute correctly, only run longer or use more energy than necessary.

**HLRS:** - Concerning the monitoring prediction's part, how do you take into consideration the predictions of the monitoring during runtime execution of the different EXCESS components?

**LIU:** The Monitoring Framework does not need to do predictions on its own, it should give us just the raw data that we need to feed into our WP1 prediction model. Of course you can extend your monitoring framework to display various metrics such as energy efficiency, energy-delay-product, or use it with a similar prediction model as in WP1 so you could then use it also stand-alone for (non-EXCESS) application analysis.

**UiT:** When the first version will be out?

**HLRS:** Now we don’t know. We have to go thought the requirements and do the time plan.

*Monitoring Framework Server*

     Allow Filtering

     Provide Visualization of data

     Export functionality (csv)

     Warm-UP period to exclude some runs

*Monitoring Framework Client*

     Plug-able/Extendable  (System Information Gatherer)

     Prediction Model using Historical Data

Allow for specific application or pid to be monitored

     Proposed calls (Start/Stop Monitoring System/pid/task, Specify Metrics and time interval)

     Automatically Triggering of remote hardware energy measurements

# State of the art

Introductory pragaragraph Introductory Introductory pragaragraph Introductory Introductory pragaragraph Introductory Introductory pragaragraph Introductory Introductory pragaragraph Introductory Introductory pragaragraph Introductory Introductory pragaragraph Introductory Introductory pragaragraph Introductory Introductory pragaragraph Introductory Introductory pragaragraph Introductory Introductory pragaragraph Introductory Introductory pragaragraph Introductory pragaragraph

## Application performance management

From Wikipedia, the free encyclopedia

In the fields of [information technology](http://en.wikipedia.org/wiki/Information_technology" \o "Information technology) and [systems management](http://en.wikipedia.org/wiki/Systems_management" \o "Systems management), **Application Performance Management**  (APM)  is the monitoring and management of performance and availability of [software](http://en.wikipedia.org/wiki/Software" \o "Software) applications. APM strives to detect and diagnose application performance problems to maintain an expected [level of service](http://en.wikipedia.org/wiki/Level_of_service" \o "Level of service). APM is the translation of [IT metrics](http://en.wikipedia.org/wiki/Performance_metric" \o "Performance metric) into business meaning (i.e., value).[[1]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-1)

## Measuring application performance

Two sets of [performance metrics](http://en.wikipedia.org/wiki/Performance_metrics" \o "Performance metrics) are closely monitored. The first set of performance metrics defines the performance experienced by end users of the application. One example of performance is average response times under peak load. The components of the set include load and response times.

* The load is the volume of transactions processed by the application, e.g., transactions per second (tps), requests per second, pages per second. Without being loaded by computer-based demands for searches, calculations, transmissions, etc., most applications are fast enough, which is why programmers may not catch performance problems during development.
* The response times are the times required for an application to respond to a user's actions at such a load.[[2]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-2)

The second set of performance metrics measures the [computational resources](http://en.wikipedia.org/wiki/Resource_%28computer_science%29" \o "Resource (computer science)) used by the application for the load, indicating whether there is adequate capacity to support the load, as well as possible locations of a performance bottleneck. Measurement of these quantities establishes an empirical performance baseline for the application. The baseline can then be used to detect changes in performance. Changes in performance can be correlated with external events and subsequently used to predict future changes in application performance.[[3]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-3)

The use of APM is common for Web applications, which lends itself best to the more detailed monitoring techniques.[[4]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-4) In addition to measuring response time for a user, response times for components of a Web application can also be monitored to help pinpoint causes of delay. There also exist [HTTP](http://en.wikipedia.org/wiki/HTTP" \o "HTTP) appliances that can decode transaction-specific [response times](http://en.wikipedia.org/wiki/Round-trip_delay_time" \o "Round-trip delay time) at the Web server layer of the application.

In their *APM Conceptual Framework*, [Gartner](http://en.wikipedia.org/wiki/Gartner) Research describes five dimensions of APM:[[5]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-5)[[6]](http://en.wikipedia.org/wiki/Application_performance_management#cite_note-6)[[7]](http://en.wikipedia.org/wiki/Application_performance_management#cite_note-7)

* End [user experience](http://en.wikipedia.org/wiki/User_experience" \o "User experience) monitoring - ([Active](http://en.wikipedia.org/wiki/Synthetic_monitoring" \o "Synthetic monitoring) and [passive](http://en.wikipedia.org/wiki/Passive_monitoring" \o "Passive monitoring))
* Application runtime architecture discovery and modeling
* User-defined transaction profiling (also called [business transaction management](http://en.wikipedia.org/wiki/Business_transaction_management" \o "Business transaction management))
* Application component monitoring
* Reporting & Application [data analytics](http://en.wikipedia.org/wiki/Data_analytics" \o "Data analytics)

## Current Issues

Since the first half of 2013, APM has entered into a period of intense competition of technology and strategy with a multiplicity of vendors and viewpoints.[[8]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-8) This has caused an upheaval in the marketplace with vendors from unrelated backgrounds (including network monitoring, systems management, application instrumentation, and web performance monitoring) to adopt messaging around APM. As a result, the term APM has become diluted and has evolved into a concept for managing application performance across many diverse computing platforms, rather than a single market.[[9]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-9)

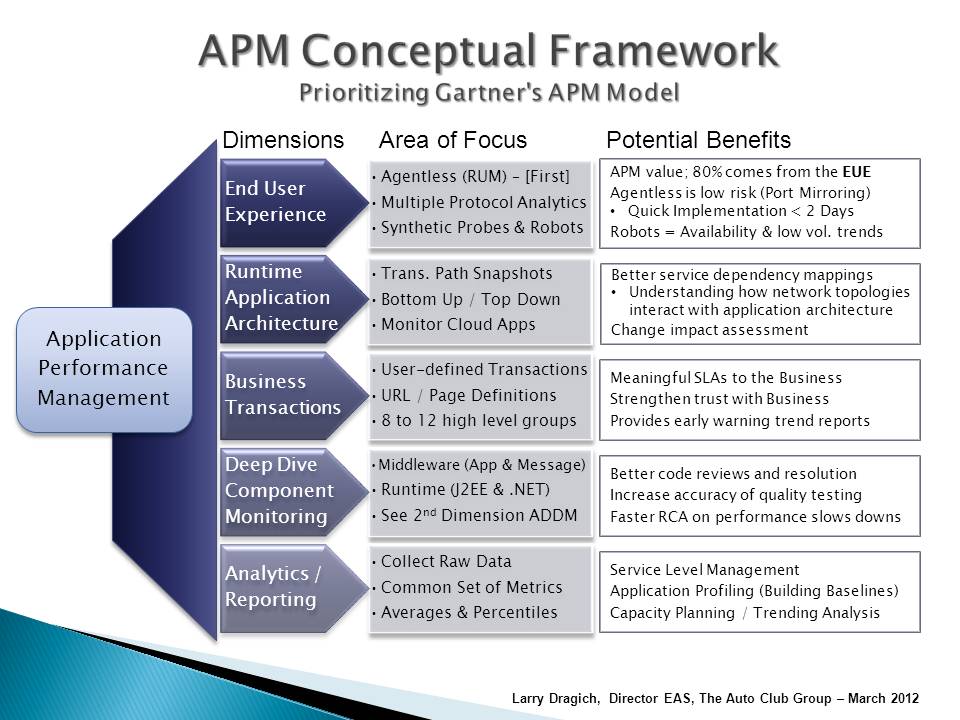
Two challenges for implementing APM are (1) it can be difficult to instrument an application to monitor application performance, especially among components of an application, and (2) applications can be [virtualized](http://en.wikipedia.org/wiki/Platform_virtualization" \o "Platform virtualization), which increases the variability of the measurements.[[10]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-10)[[11]](http://en.wikipedia.org/wiki/Application_performance_management#cite_note-11) Distributed, virtual and [cloud-based](http://en.wikipedia.org/wiki/Cloud_computing" \o "Cloud computing) applications pose a unique challenge for application performance monitoring because most of the key system components are no longer hosted on a single machine. Each function is now likely to have been designed as an Internet service that runs on multiple virtualized systems. The applications themselves are very likely to be moving from one system to another to meet service-level objectives and deal with momentary outages.[[12]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-12)

## The APM Conceptual Framework

Applications themselves are becoming increasingly difficult to manage as they move toward highly-distributed, multi-tier, multi-element constructs that in many cases rely on application development frameworks such as. NET or Java.[[13]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-13) The APM Conceptual Framework was designed to help prioritize an approach on what to focus on first for a quick implementation and overall understanding of the five-dimensional APM model. The framework slide outlines three areas of focus for each dimension and describes their potential benefits. These areas are referenced as "*Primary*" below, with the lower priority dimensions referenced as "*Secondary.* "[[14]](http://en.wikipedia.org/wiki/Application_performance_management#cite_note-14)

### End User Experience – (Primary)

Measuring the transit of traffic from user request to data and back again is part of capturing the end-user-experience (EUE).[[15]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-15) The outcome of this measuring is referred to as Real-time Application monitoring (aka Top Down monitoring), which has two components, Passive and Active. [Passive monitoring](http://en.wikipedia.org/wiki/Passive_monitoring" \o "Passive monitoring) is usually an agentless appliance implemented using [network port mirroring](http://en.wikipedia.org/wiki/Port_mirroring" \o "Port mirroring). A key feature to consider in this solution is the ability to support multiple protocol analytics (e.g., XML, SQL, PHP) since most companies have more than just web-based applications to support. [Active monitoring](http://en.wikipedia.org/wiki/Synthetic_monitoring" \o "Synthetic monitoring), on the other hand, consists of synthetic probes and web robots predefined to report system availability and business transactions. Active monitoring is a good complement to passive monitoring; together, these two components help provide visibility into application health during off peak hours when transaction volume is low.

[](http://en.wikipedia.org/wiki/File:APM_Conceptual_Framework.jpg)

[http://bits.wikimedia.org/static-1.23wmf21/skins/common/images/magnify-clip.png](http://en.wikipedia.org/wiki/File:APM_Conceptual_Framework.jpg)

This slide outlines three *areas of focus* for each dimension and describes their potential benefits.

*User Experience Management* (UEM) is a subcategory that emerged from the EUE dimension to monitor the behavioral context of the user. UEM, as practiced today, goes beyond availability to capture latencies and inconsistencies as human beings interact with applications and other services.[[16]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-16) UEM is usually agent-based and may include JavaScript injection to monitor at the end user device. UEM is considered another facet of Real-time Application monitoring.

### Runtime Application Architecture (Secondary)

Application Discovery and Dependency Mapping (ADDM) solutions exist to automate the process of mapping transactions and applications to underlying infrastructure components.[[17]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-17) When preparing to implement a runtime application architecture, it is necessary to ensure that up/down monitoring is in place for all nodes and servers within the environment (aka, bottom-up monitoring). This helps lay the foundation for event correlation and provides the basis for a general understanding on how network topologies interact with application architectures.

### Business Transaction (Primary)

Focus on user-defined transactions or the URL page definitions that have some meaning to the business community. For example, if there are 200 to 300 unique page definitions for a given application, group them together into 8-12 high-level categories. This allows for meaningful SLA reports, and provides trending information on application performance from a business perspective: start with broad categories and refine them over time. For a deeper understanding see [Business Transaction Management](http://en.wikipedia.org/wiki/Business_Transaction_Management" \o "Business Transaction Management).

### Deep Dive Component Monitoring (Secondary)

Deep Dive Component Monitoring (DDCM) requires an agent install and is generally targeted in the [middleware](http://en.wikipedia.org/wiki/Middleware" \o "Middleware) space focusing on the web, application, and messaging servers. It should provide a real-time view of the [J2EE](http://en.wikipedia.org/wiki/J2EE" \o "J2EE) and [.NET](http://en.wikipedia.org/wiki/.NET_Framework" \o ".NET Framework) stacks, tying them back to the user-defined business transactions. A robust solution shows a clear path from a code execution standpoint (e.g., Spring, Struts, etc.) to the URL rendered and finally to the user request. Since DDCM is closely related to the second dimension in the APM model, most products in this space also provide application discovery dependency mapping (ADDM) as part of their broader solution.

### Analytics/Reporting (Primary)

It is important to arrive at a common set of metrics to collect and report on for each application, then standardize on a common view on how to present the application performance data. Collecting raw data from the other tool sets across the APM model provides flexibility in application reporting. This allows for answering a wide variety of performance questions as they arise, despite the different platforms each application may be running on. Too much information is overwhelming. That is why it is important to keep reports simple or they won’t be used.[[18]](http://en.wikipedia.org/wiki/Application_performance_management" \l "cite_note-18)

It is import that we identify where our monitoring framework belongs before being able to select the state of the art. Maybe this part (APM) should be moved after the requirements.

## Nagios[[1]](#footnote-1)

Nagios is an [open source](http://en.wikipedia.org/wiki/Open_source) [computer](http://en.wikipedia.org/wiki/Computer) [system monitoring](http://en.wikipedia.org/wiki/System_monitor), [network monitoring](http://en.wikipedia.org/wiki/Network_monitoring) and infrastructure monitoring [software application](http://en.wikipedia.org/wiki/Software_application). Nagios offers monitoring and alerting services for servers, switches, applications, and services. It alerts the users when things go wrong and alerts them a second time when the problem has been resolved.

## Kieker[[2]](#footnote-2)

## Zabbix[[3]](#footnote-3)

## OpenNMS[[4]](#footnote-4)

## Ganglia[[5]](#footnote-5)

Ganglia is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on thousands of clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes.

# High level architecture

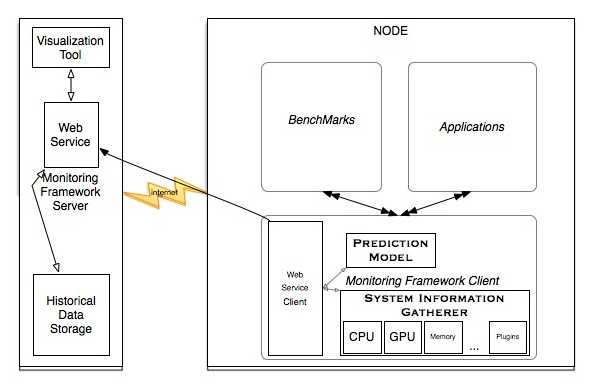


Figure 1: Excess monitoring Framework

This architecture for the Monitoring framework is based on the “producer-consumer” concept, which is widely applied in the monitoring systems. We propose the monitoring framework that consists of two main components. The first one is the Client, which will run on each node of the cluster; the second one is the server, which will run on the front end, as shown in Figure 1. The Client will be able to gather system information of the metrics of interest and send this information to the server at a time interval defined by the user. The Server will collect the information being sent by the nodes and will offer functionalities to query and visulalize the collected data.

“Client” is actually called “Node” in the figure. Many boxes of the figures remain unexplained, including their interconnection/work together

# Implementation methodology

## Agile principles

The Agile Manifesto is based on twelve principles:[[8]](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-manifestoprinciples-8)

1. Customer satisfaction by rapid delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Working software is the principal measure of progress
5. Sustainable development, able to maintain a constant pace
6. Close, daily cooperation between business people and developers
7. Face-to-face conversation is the best form of communication (co-location)
8. Projects are built around motivated individuals, who should be trusted
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Self-organizing teams
12. Regular adaptation to changing circumstances

## Agile Overview

Iterative, incremental and evolutionary

Agile methods break tasks into small increments with minimal planning and do not directly involve long-term planning. Iterations are short time frames ([timeboxes](http://en.wikipedia.org/wiki/Timeboxing)) that typically last from one to four weeks. Each iteration involves a [cross-functional team](http://en.wikipedia.org/wiki/Cross-functional_team) working in all functions: planning, [requirements analysis](http://en.wikipedia.org/wiki/Requirements_analysis), [design](http://en.wikipedia.org/wiki/Software_design), [coding](http://en.wikipedia.org/wiki/Computer_programming), [unit testing](http://en.wikipedia.org/wiki/Unit_testing), and [acceptance testing](http://en.wikipedia.org/wiki/Acceptance_testing). At the end of the iteration a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly. An iteration might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal [bugs](http://en.wikipedia.org/wiki/Software_bug)) at the end of each iteration.[[10]](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-embracing_change-10) Multiple iterations might be required to release a product or new features.

Efficient and face-to-face communication

No matter what development disciplines are required, each agile team will contain a [customer representative](http://en.wikipedia.org/wiki/Customer_representative), e.g. Product Owner in Scrum. This person is appointed by stakeholders to act on their behalf[[11]](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-11) and makes a personal commitment to being available for developers to answer mid-iteration questions. At the end of each iteration, stakeholders and the customer representative review progress and re-evaluate priorities with a view to optimizing the [return on investment](http://en.wikipedia.org/wiki/Rate_of_return) (ROI) and ensuring alignment with customer needs and company goals.

In agile software development, an information radiator is a (normally large) physical display located prominently in an office, where passers-by can see it. It presents an up-to-date summary of the status of a software project or other product.[[12]](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Cockburn.2C_Information_radiator-12)[[13]](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Ambler-13) The name was coined by [Alistair Cockburn](http://en.wikipedia.org/wiki/Alistair_Cockburn), and described in his 2002 book Agile Software Development.[[13]](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Ambler-13) A [build light indicator](http://en.wikipedia.org/wiki/Build_light_indicator) may be used to inform a team about the current status of their project.

Very short feedback loop and adaptation cycle

A common characteristic of agile development are daily status meetings or "[stand-ups](http://en.wikipedia.org/wiki/Stand-up_meeting)", e.g. Daily Scrum (Meeting). In a brief session, team members report to each other what they did the previous day, what they intend to do today, and what their roadblocks are.

Quality focus

Specific tools and techniques, such as [continuous integration](http://en.wikipedia.org/wiki/Continuous_integration), automated [unit testing](http://en.wikipedia.org/wiki/XUnit), [pair programming](http://en.wikipedia.org/wiki/Pair_programming), [test-driven development](http://en.wikipedia.org/wiki/Test-driven_development), [design patterns](http://en.wikipedia.org/wiki/Software_design_pattern), [domain-driven design](http://en.wikipedia.org/wiki/Domain-driven_design), [code refactoring](http://en.wikipedia.org/wiki/Code_refactoring) and other techniques are often used to improve quality and enhance project agility.

Add a paragraph here stating how we used the above mentioned approach for the implementation of the monitoring framework and why we chose it.

# Technologies selected

## Client Side

### C library

Minimum interval 1 sec. We are waiting for more information from WP1 to see what is the delay necessary, because sending real time data can be intrusive. The initial idea is to buffer data and then send it to the server. In order to do online training and prediction the size of the buffer has to be as small as possible. We have to find the trade-off between accuracy and intrusiveness of the monitoring framework.

### Libcurl

## Predictor

The prediction model has a number of parameters which are trained from the observations using lineal regression to estimate the parameters.

The best way to do that is to do an online training not an offline training. The idea is: during execution there can be communication between the runtime system and the monitoring framework, going back to the server and getting update of the latest info that exist, and using the actual execution raw data only to have details of what is going to happen.

- It should allow to collect measurements from both performance/energy counters and, where available, additional hardware measurement equipment. If additional hardware measurements are available, it should automatically calibrate the counter-based measurements with real data. If only performance counters are available, it should still be usable, with possibly lower accuracy.

## Server Side

### Node.js

### ElasticSearch

# Initial implementation results

### Walkthrough the screens(UI)

### Backend details

# Possible ways of extending

Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text …

# References

[1] Ref…

[2] Ref…

# Glossary

|  |  |
| --- | --- |
| **ACRONYM1** | description |
| **ACRONYM2** | description |

1. http://www.nagios.org [↑](#footnote-ref-1)
2. http://kieker-monitoring.net [↑](#footnote-ref-2)
3. http://www.zabbix.com [↑](#footnote-ref-3)
4. http://www.opennms.org [↑](#footnote-ref-4)
5. http://ganglia.sourceforge.net [↑](#footnote-ref-5)