www.czasopisma.pan.pl PAN POLSKA AKADEMIA NAUK www.journals.pan.pl

*Metrol. Meas. Syst.*, Vol. XIX (2012), No. 2, pp. 177-190.

PAN POLSKA AKADEMIA NAUK

METROLOGY AND MEASUREMENT SYSTEMS

Index 330930, ISSN 0860-8229

*www.metrology.pg.gda.pl*

M&MS

# EVALUATION OF MULTIMEDIA APPLICATIONS IN A CLUSTER-ORIENTED ENVIRONMENT

**Paweł Czarnul, Tomasz Dziubich, Henryk Krawczyk**

*Gdansk University of Technology, Faculty of Electronics Telecommunications and Informatics, Department of Computer Systems Architecture, Narutowicza 12/11, 80-233 Gdansk, Poland (✉ pczarnul@eti.pg.gda.pl, dziubich@eti.pg.gda.pl, hkrawk@eti.pg.gda.pl, +48 58 347 2863)*

**Abstract**

In the age of Information and Communication Technology (ICT), Web and the Internet have changed significantly the way applications are developed, deployed and used. One of recent trends is modern design of web-applications based on SOA. This process is based on the composition of existing web services into a single scenario from the point of view of a particular user or client. This allows IT companies to shorten the product-time to market process. On the other hand, it raises questions about the quality of the application, trade-offs between quality factors and attributes and measurements of these. Services are usually hosted and executed in an environment managed by its provider that assures the quality attributes such as availability or throughput. Therefore, in this paper an attempt has been made to perform quality measurements towards the creation of efficient, dependable and user-oriented Web applications. First, the process of designing service-based applications is described. Next, metrics for subsequent measurements of efficiency, dependability and usability of distributed applications are presented. These metrics will assess the efforts and trade-offs in a Web-based application development. As examples, we describe a pair of multimedia applications which we have developed in our department and executed in a cluster-based environment. One of them runs in the BeesyCluster middleware and the second one in the Kaskada platform. For these applications we present results of measurements and conclude about relations between quality attributes in the presented application development model. This knowledge can be used to reason about such relations for new similar applications and be used in rapid and quality development of the latter.

Keywords: quality measurements, software quality, quality model and measures, parallel computing, distributed middleware, multimedia applications.



## 1. Introduction

Nowadays the growing complexity of computer systems is forcing new approaches to software development. One of recent trends is modern design of web applications based on SOA [1]. This allows IT companies to shorten the product-time to market process. But on the other side, it also implies the need for increasing of computing power and high flexibility of components. This is an especially important aspect in applications that process huge-volume data, i.e. multimedia applications. The increase in the processing performance is obtained by deploying computationally expensive application modules in a cluster–oriented environment, while high flexibility is maintained by standardizing ways of communication between modules/services and the introduction of components reuse.

Traditionally, development of high-performance applications requires knowledge and experience of low-level solutions such as parallel MPI-based programming for image recognition, processing of multimedia streams (variant A in Fig. 1). A better solution is to use a middleware such as e.g. IBM WebSphere MQ (Variant B). Instead, in our department we have created two solutions Kaskada [2] and BeesyCluster [3] for easy and fast building of

Article history: received on Feb. 6, 2012; accepted on Apr. 27, 2012; available online on May 18, 2012; DOI: 10.2478/v10178-012-0016-9.

complex applications from services and ready-to-use blocks (variant C in Fig. 1). The process of developing applications in such an environment is thus reduced to the following steps:

- develop the required algorithms which have not yet been implemented or use known algorithms from available libraries,
- transform algorithms into independent tasks/services according to SOA interfaces,
- preparation of service scenarios possibly using available patterns (e.g. for recognition of detected objects or events) and construction of the user interface.

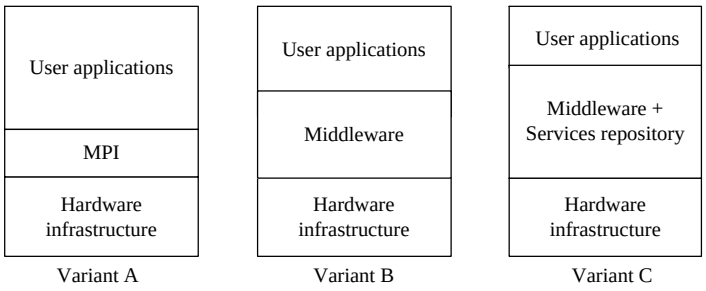


Fig. 1. Alternatives to development of applications for cluster-oriented environment.

Among the components of these platforms, useful utilities for designing user applications include: the library of algorithms, a repository of offered web services and a set of scenarios defining the behaviour of the built user applications. The developed algorithms belong to the following categories: object tracking, object detection, event recognition, estimating the number of existing objects, identification and location of sound sources (general algorithms), comprises monitoring changes in the space of process control processing, allocation of tasks, management messages (system algorithms). Many of these algorithms operate on either static input data or data streams provided to the algorithms at runtime.

For the full acceptance of such user applications, an evaluation of their quality level is also important, including the study of interdependence of quality parameters [4]. For example, the cost and effort of creation of services and the application results in corresponding service and application reliability, cost and execution time.

The rest of the paper is organized as follows. Section 2 presents platforms BeesyCluster and Kaskada that serve as environments for quality measurements of multimedia applications. Section 3 presents a quality model along with attributes and metrics for the two platforms. Section 4 shows results for real applications while Section 5 concludes the work.

## 2. Platforms supporting cluster-oriented computing

### *2.1. BeesyCluster and its model of a complex multimedia application*

BeesyCluster[1] [5, 6] is a middleware that allows many users to access, share and integrate distributed resources and services. Users access resources such as commodity servers and HPC clusters via single sign-on and individual accounts in BeesyCluster which can be bound to one or more system accounts on the servers and clusters. The platform supports, among others, an integrated environment that allows to:

- manage, develop and compile codes on multiple servers/clusters,
- apply versioning,
- launch and queue applications on servers/clusters with graphical interfaces through a Web browser hiding details of queuing systems such as PBS, LSF, LoadLeveler, etc.,

[1] https://lab527.eti.pg.gda.pl:10030/ek/Main

*Metrol. Meas. Syst.*, Vol. XIX (2012), No. 2, pp. 177-190.

- monitor states of distributed clusters graphically,
- develop applications in groups with task assignment, sharing files among users and interactive shared white boards and chat,
- become a provider and instantly publish own or existing applications as services right from the server/cluster on which these have been compiled,
- assign cost and privileges to published services on a per - user or per - group basis; the provider can be associated with reputation or reliability of their services,
- create, manage, optimize and run complex workflow applications built out of either own services or services published by others and made available to the given user. This allows integration of highly distributed services, optimization of service selection using various QoS parameters. One centralized Java EE engine [5] and one distributed agent-base engine [6] were developed for efficient running of such workflow applications. Such workflows can be treated as reusable templates, static and dynamic optimization of service selection and determination of optimal data flows [7] to minimize QoS including execution time, cost and as proposed in this work reliability.

A workflow is represented by an acyclic directed graph in which nodes correspond to tasks and edges to time dependencies between tasks [5, 6]. A set of services $S_i$ is assigned to each task $t_i$. $S_i$ contains services $s_{ij}$, each of which is capable of executing task $t_i$. The basic parameters [5, 6] of the task are cost $c_{ij}$ and execution time $t_{ij}$. For each task one service needs to be selected to perform the task. Data size processed by task $t_i$ and the selected service $s_{ij}$ is denoted as $d_{ij}$. One of possible optimization goals is minimization of the workflow execution time *min* $t_{workflow}$ with a bound on the cost of selected services

$$\sum c_{ij} d_{ij} \leqslant C_{max}$$

or minimization of a linear combination of cost and time $\sum c_{ij} d_{ij} + a t_{workflow}$.

Optimization of scheduling workflow applications, especially in cluster and grid environments has been studied widely in the literature [8-12]. It should be noted that BeesyCluster allows to extend easily the description of a service with more quality metrics easily and subsequent incorporation of those in optimization. This work proposes how to extend the models presented by one of the authors in [5, 6] by introducing reliability of both the infrastructure as well as the services.

### *2.2. Kaskada platform*

Kaskada platform [2], developed within the Mayday 2012 project[2], is a novel approach in the field of application development for the cluster environment. Kaskada is a universal runtime platform for algorithms processing multimedia streams, e.g. videos and sound recordings. The platform operates in the Galera cluster system [13], using its enormous computing power and making it available for executed algorithms. It is a perfect solution for algorithms presenting high demand on computational power, examples of which are image recognition algorithms supporting medical endoscopic examinations of the gastrointestinal tract. At the time of rapid development of high power computers, performing computation at low level, and many arising challenges associated with more abstract computer vision tasks, such as analyzing videos from surveillance cameras or analysis of medical images, there appears to be a need for a solution effectively connecting the two areas, enabling successful construction and execution of stream-processing algorithms in the environment of a supercomputer.

The platform supports an application on three levels of functionality:

[2] http://mayday2012.gda.pl

- Stream level – the goal is to maintain a massive load of multimedia data. The functionality includes playing, stopping, archiving, replaying, distributing, multiplying of multimedia streams and load balance;
- Service level – the goal is processing user/application requests. The functionality includes invoking, finishing, monitoring, killing, assigning of user tasks;
- Event level – the goal is to provide means to communicate the processing state to the user/application. The basic functionality is generating, storing, distributing, filtering, relaying of output messages.

Except being a powerful execution environment for time-consuming algorithms, Kaskada also provides a universal external interface in the form of automatically created web services, enabling launching algorithms from remote applications, e.g. from the doctor's office. Kaskada is also a framework facilitating the construction of stream algorithms. The platform performs all video decoding tasks, passing raw frames to the algorithm. Also, extensive communication mechanisms are provided by the platform, enabling construction of highly parallelized, distributed algorithms in the form of computational services engaging multiple processors.

## 3. Application quality, metrics, measurement techniques

### *3.1. Quality attributes and metrics in BeesyCluster*

Execution of complex workflow applications in BeesyCluster was designed to allow to control and find desired balance among the following quality attributes according to the quality model presented in [5]:

- *performance* – achieving high parallel *efficiency* and *scalability* of processing multimedia data is possible by engaging several services installed on various clusters and nodes to process data in parallel. *Scalability* and speed-up is determined by: the ratio of the computational time of the services compared to the communication time of transferring data between services and the overhead of the underlying execution engine [5, 6]. It can be affected by *granularity* of processing and data streaming; the workflow execution engine in BeesyCluster allows two processing modes of each workflow node: streaming and non-streaming and allows mixing nodes of the two types in one workflow. Furthermore, to optimize the workflow execution time, it can automatically pack a large number of small files into an archive to be sent to following workflow nodes to minimize the communication latency. *Efficiency* of service implementations should be assured at the service level; for example, a service based on application *convert -normalize* from ImageMagick referring to disk space often could use faster scratch space on the cluster on which it was deployed. This can decrease its running time by a factor of 10. For long-running workflows the overhead of the workflow execution engine is small [6].
- *dependability* – defined in particular by:
  *reliability: Reliability* of executing a complex task is assured by running it in the Java EE environment supporting transactions; let $r_{ij}$ denote the reliability of the infrastructure on which service $s_{ij}$ runs and also the reliability of the connection to the service. *Reliability* of services/providers is defined as follows. Let $R_{ij}$ denote how reliable and precise the results of the service are; a higher $R_{ij}$ will most likely result in a higher execution time of the service or a greater cost because of the need for more powerful computers.
  *error tolerance* - the model allows to continue running complex workflow applications even if failures of particular services have occurred [6]; the execution engine automatically reselects services for the remaining part of the workflow considering available services and previous selections;

*Metrol. Meas. Syst.*, Vol. XIX (2012), No. 2, pp. 177-190.

*security* – made sure by secure links and authorization between the client and BeesyCluster servers and from BeesyCluster to remote servers/clusters.

– *user satisfaction/usability* which can be defined by:
*ease of use* and *learnability* – possible through easy creation and running using WWW and Web Service interfaces; it was possible to define and run workflow applications by students of Architectures of Internet Services from one to a few hours of work including learning the environment. Non-specialists could use basic BeesyCluster functions listed in Section 2.1 after just 2 hours of training.
*productivity* – the system allows to define and reuse complex workflow applications. It allows rapid development and definition of either performance- or reliability-oriented applications once and running these many times with adaptable runs and runtime optimization.

Regarding the previously proposed optimization model [5, 6], the reliability parameters can be updated at runtime by the underlying workflow execution engine based on the history. It is possible to:

1. estimate the effective service execution time based on the learnt reliability of service infrastructure i.e.:

$$t_{ij} = \left(1/r_{ij}\right) \cdot t_{tj}^{failure-free}.$$

where $t_{tj}^{failure-free}$ denotes the execution time of service $s_{ij}$. This is not a problem even in the linear integer problem formulation since $r_{ij}$ is updated at runtime but can be considered a constant during optimization.

2. use the history and digital filters to estimate the effective running time of a service (including potential failures) e.g.:

$$\left(1/10\right)\sum_{x=1}^{10} t_{ij}^{x},$$

where $t_{ij}^{x}$ denotes the *x-th* last running time of service $s_{ij}$ or

$$t_{ij} = \left(1/55\right)\sum_{x=1}^{10} (11-x)t_{ij}^{x},$$

to assign higher weights to more recent measurements of execution time.

The end user may specify the minimum reliability of results returned by the service while minimizing the execution time of the whole workflow, e.g.: *min* $t_{workflow}$ with constraints on the reliability of results and costs:

$$\forall_{selected\ ij} R_{ij} \geqslant R_{min}, \sum c_{ij}\, d_{ij} \leqslant C_{max}.$$

All constraints can be incorporated into the genetic algorithm proposed in [6]. Namely, the random selection of services for a particular solution (chromosome) has to consider only the available services that have reliabilities higher than the given threshold $R_{min}$.

### *3.2. Quality attributes and metrics in Kaskada*

Quality assessment was performed for the application of the medical recommendations. The most important problems of computer-aided diagnosis include: reducing the time of diagnosis, expanding the range of algorithms for medical recommendations and increasing the efficiency of endoscopic image recognition. In the paper we concentrate on the first problem. To resolve it, we have constructed application ERS 2012 which consists of three major

components: medical examination station, frontend server and Kaskada platform [14]. The physician can upload a movie from an endoscope or a wireless capsule endoscopy using Media Streaming Server to Frontend Server (FES). FES calls the appropriate scenario that is located on the Kaskada platform. In ERS 2012, it is possible to transmit a movie from an endoscope in real-time using the Frame Grabber module. But the well-known sequential algorithms are not enough to perform efficient online recommendation. Several ways of paralleling stream algorithms are enabled by the Kaskada platform. The parallelization techniques described later in the article were applied with presented algorithms. In the Kaskada platform, stream processing can be simplistically illustrated as shown in Fig. 2.

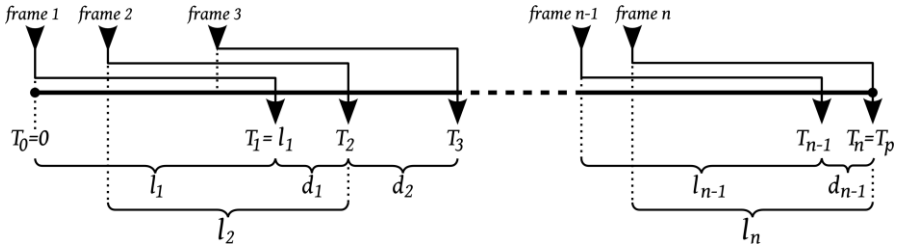


Fig. 2. Stream processing in the Kaskada platform.

$n$ – number of frames in the whole stream
$T_0$ – time at which the first frame arrives to the system, fixed to 0
$T_i$ – time at which i-th frame's report is available
$T_p$ – time of the whole video processing, $T_p = T_n - T_0$
$l_i$ – processing time of i-th frame
$d_i$ – time interval between two succeeding reports, $d_i = T_{i+1} - T_i$

For this application we propose the following evaluation metrics:

– *performance* – test of performance of a stream-processing application consisted of throughput measurement. The throughput of the pipeline system is the maximum amount of data that the system can process in a given time; videos were processed separately, with maximal input frame rate (>100 fps). Every video was processed 100 times. Finally, the average throughput $H$ and $\sigma_H$ (standard deviation) were calculated. While processing the video, temporal throughput values $h_i$ have been calculated using the relationship: $h_i = 1/d_i$. After processing the whole video, the average throughput value $H$ has been computed as the mean value of the temporal values weighted by their time:

$$H = \frac{n-1}{\sum_{i=1}^{n-1} d_i},$$

which gives the ratio of the number of frames to the amount of time needed to process them – the well-known FPS (frames per second) measure. Finally, to acquire the standard deviation of the temporal throughput, its variance has been calculated using the weighted formula

$$\sigma_H^2 = \frac{\sum_{i=1}^{n-1} (h_i - H)^2 d_i}{\sum_{i=1}^{n-1} d_i},$$

– *dependability* – is defined as the number of lost frames to the number of all analyzed frames. This corresponds to the *latency measurement*. The latency is the time between the arrival of a video frame at the system input and the time at which the detection report is available at the system output; videos were processed separately, with their original frame rate (25 fps) for all the versions of the algorithm which proved to provide a sufficient

*Metrol. Meas. Syst.*, Vol. XIX (2012), No. 2, pp. 177-190.

throughput. Other configurations were skipped as being not suitable for real life use. Every video was processed 10 times. Finally, the average $L$ and the standard deviation $\sigma_L$ of the latency were calculated using the simple formulas:

$$L = \frac{\sum_{i=1}^{n} l_i}{n} \quad \text{and} \quad \sigma_L^2 = \frac{\sum_{i=1}^{n} (l_i - L)^2}{n}.$$

For each case, required parameters were measured: the processing rate, represented by the average number of frames processed per second, and the delay, being a single frame processing time. Tests were performed on 5 video sequences of 720x576 resolution, twice for each sequence, giving a total of 10 measurements for each case. Averaged results were presented on charts in chapter 4.2. Standard deviations of each sample were marked in the form of vertical bars at the average values.

– *user satisfaction/usability* which can be defined by an average measure of the diagnostic time and matching abnormal regions. For usability testing, the examination data and evaluation procedures were implemented with the assistance of medical doctors in the Medical University of Gdansk, Poland. To ensure unbiased evaluations, the experiments were set up under conditions of normal diagnostic procedures. MedEye – user interface of the ERS2012 system – was installed on a medical examination station to present the proposed method, which also supports common functions such as the capturing of abnormal regions, the changing of display modes, the adjustment of skill levels, and functions to navigate and scanning/browsing frame-by-frame.
Twenty sequences from patients were selected. The total length of these sequences was 307 minutes. The evaluations were carried out by two medical doctors. They were asked to independently find and capture suspicious regions. The time codes of abnormal regions as well as the events/activities of the medical doctors during the diagnostic procedures were logged. For assessment of the capacity and performance, these data were then analyzed and inspected as described below.
*Average measure of the diagnostic time.* To explore in detail the diagnostic time for each evaluation section, the time code data at the moment of each start/stop action was analyzed. In addition, frame-by-frame scanning to finding abnormal regions was also inspected. The diagnostic time is the total of the following two components:
  – Playing time: the total duration that each medical doctor played the sequences continuously, without actions such as jumping, scanning, or frame navigation.
  – Scanning/Browsing time: the total time for browsing or frame-by-frame scanning to verify abnormal regions.

Thus, the main difference between this method and other methods [15] is that the reading time details are inspected by two separate components, and this helps one to better understand not only the time for viewing a sequence but also the time used for seeking abnormal regions.
*Matching abnormal regions captured*. In the experiment, the medical doctors were asked to capture abnormal regions independently. Then, knowing the degree to which the abnormal region capture precised were accurate and complete would allow the performance of the method to be evaluated, a routine for checking the relevant findings was therefore implemented after the evaluations of the medical doctors. Previous studies showed only the total diagnostic time without information regarding the verification of abnormal regions detected. To check abnormal regions, we compared the results of two medical doctors, and in cases of discrepancies a third gastroenterologist made the final decision.

## 4. Quality assessment for BeesyCluster and Kaskada applications

### *4.1. BC methods and results for performance evaluation*

As an example, the workflow application presented in Fig. 3 was used. This allows parallel processing of RAW camera images given as input. For input, 80 RAW images of around 15 MB each (Pentax's PEF format) were used. Eight paths, each of which can be performed in parallel were defined. Each path consists of three steps represented by successive workflow tasks: conversion of a RAW image to a 16-bit TIFF, normalization of the TIFF and conversion to JPG, resizing and reduction of quality and file size. Finally, a web album is created out of the images processed by the parallel paths.

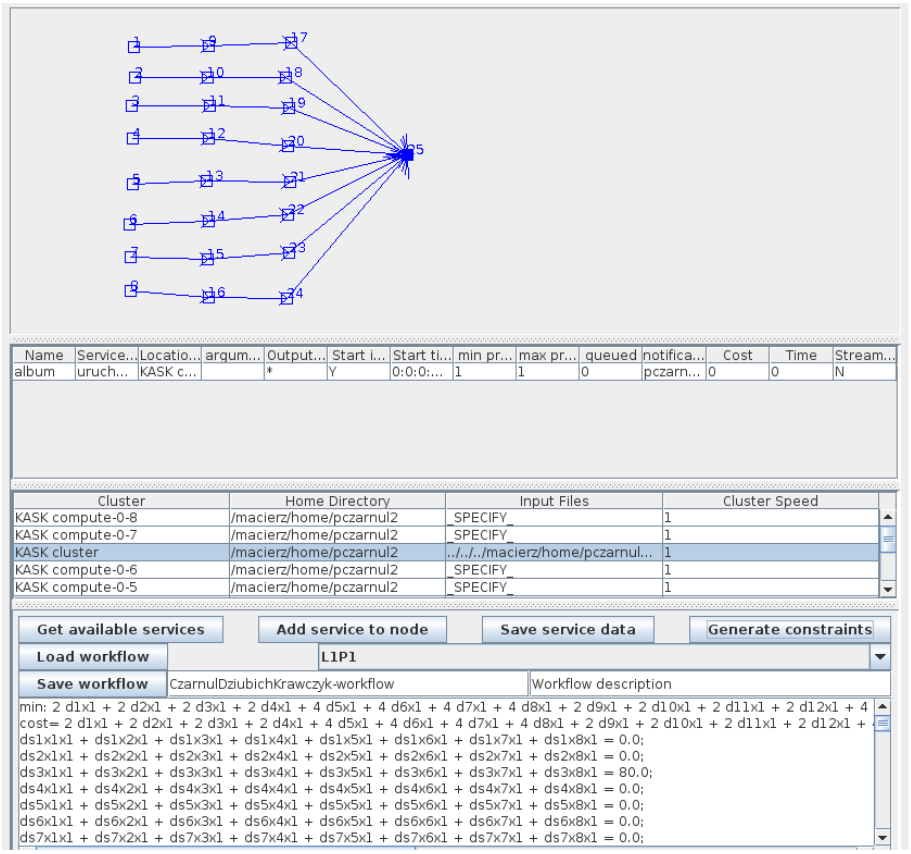


Fig. 3. A workflow application for processing digital images by 8 parallel paths.

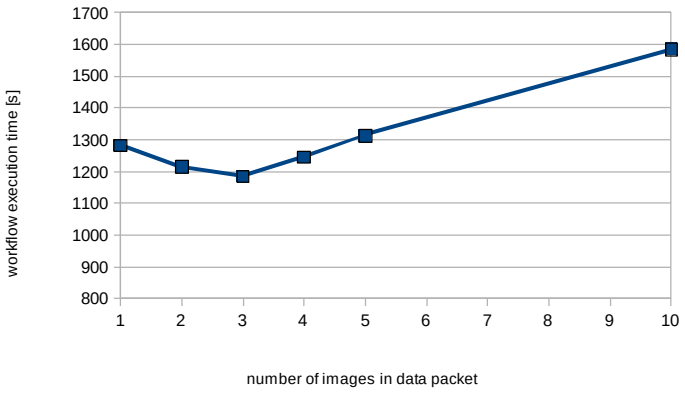


Fig. 4. Impact of granularity on the workflow execution time.

*Metrol. Meas. Syst.*, Vol. XIX (2012), No. 2, pp. 177-190.

Fig. 4 presents the impact of granularity on the workflow execution time, assuming that data packets of a certain size (in this case the number of images in a packet) are passed from one node to the next one as soon as it is available. This in fact implements parallel pipelining by services executing the pipelines in parallel. The maximum number of images allowed to be stored in a node at a time is 10 and the experiment aimed at minimization of $t_{workflow}$.

Furthermore, Fig. 5 presents execution times for a processing images workflow in which services for a particular path were installed on separate or the same cluster. In the latter case, 160 input images are processed in the non-streaming fashion (following [5]) while in the first case processing of 320 images (with increased efficiency of using a scratch space for processing) was performed in the streaming mode via each service on a separate cluster using HyperThreading for the 16 path configuration. Using Hyperthreading along with increased communication costs due to many more clusters communicating between each other limits scalability. On the other hand it increases flexibility since it corresponds to a scenario with services offered by various providers. The goal was to minimize $t_{workflow}$. Cluster nodes with two dual-core Intel Xeon 2.8 GHz processors with Hyperthreading with 4 GB RAM were used.

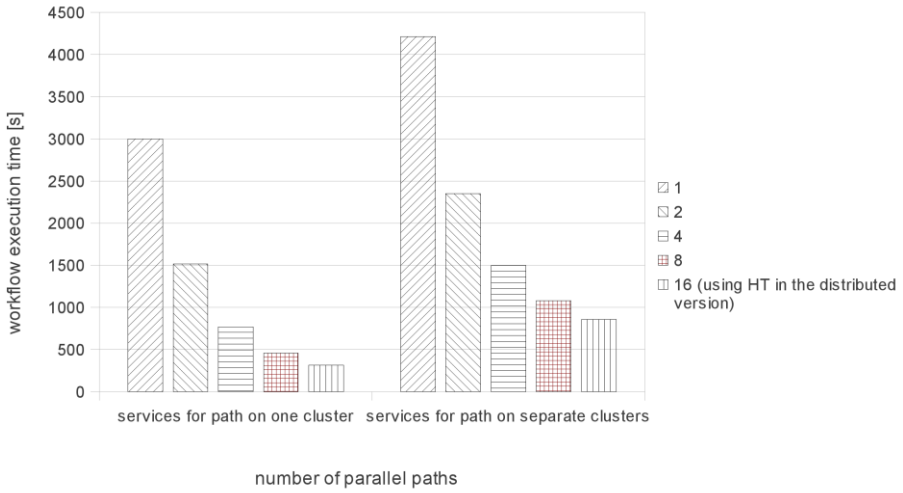


Fig. 5. Workflow execution times [s] vs number of parallel paths.

Concerning the reliability, a workflow similar to that shown in Fig. 3 was used for minimization of

$$\sum c_{ij} d_{ij} + 10 t_{workflow}.$$

It consists of 9 paths with 3 groups of 3 paths each. Particular groups have services with the following execution time/reliability/cost parameters: 2/3/5, 4/2/4, 8/1/3. Generally the cost is higher for faster services. In this experiment it is also assumed that reliability of faster services is higher because of using parallel machines to run them although it does not have to be the case generally. Limiting the required reliability results in fewer parallel paths from being selected and thus higher execution times. Table 1 shows the results, assuming minimum reliability of any selected service to 1, 2 or 3. . Configuration X requires minimum reliability of services selected at least equal to X. Correspondingly, configuration 1 results in more and cheaper services available while configuration 3 in fewer and more expensive services meeting this requirement.

Table 1. The relationship between the minimum required service reliability and the cost and execution time of a workflow application in BeesyCluster.

| Configuration/minimum reliability of services selected | 1 | 2 | 3 |
|---|---|---|---|
| Execution time [s] | 1651 | 1751 | 2085 |
| Cost | 1215 | 1872 | 2025 |

### *4.2. Kaskada methods and results for performance and usability evaluation*

#### *4.2.1. Performance*

The execution environment of the platform ensured that each separate thread in every service performing computations had an exclusive use of one processing core. The Kaskada platform enables distributing consecutive frames of a video stream to different processors. In this way, multiple frames can be processed concurrently, which can be denoted as a frame-level parallelization. Fig. 6 presents a sample service implementing this case. A single node is designated for distributing the frames among computational tasks organized in a layer, performing as separate instances of the algorithm. The last node in this scenario collects computed results and generates the output of the service.

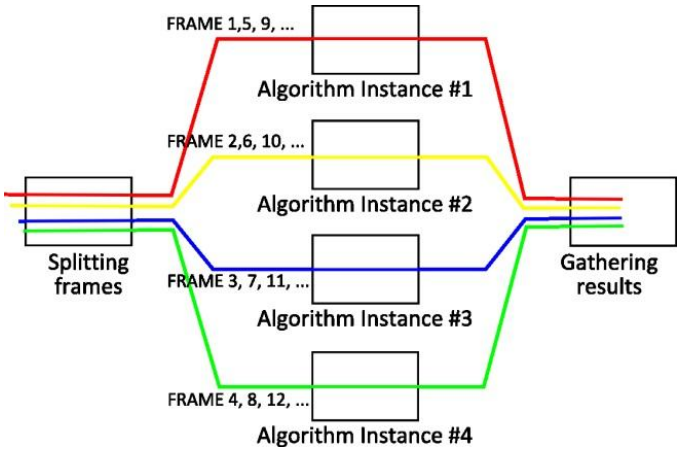


Fig. 6. Distributing a frame sequence to 4 computational tasks.

The advantages of this technique are high versatility and simplicity of implementation, since the mechanism is independent of the parallelized algorithm, provided that dependencies between frames are not considered. Moreover, this solution allows the processors power to effectively utilize and significantly reduce the overall processing time. Extension to any number of processors is possible providing high scalability. Unfortunately, the mechanism in no way reduces the processing time of a single frame, so that the delay remains unchanged comparing to the sequential processing. Fig. 7 presents results achieved accordingly to the number of processors in the processing layer. The experiments were performed for the four diagnostic algorithms: BaopuLi1 [16], Kodogiannis1 [17], Magoulas1 [18] and Magoulas2 [19]. Each of the algorithms achieved stable processing rate growth. As expected, the delay remained unchanged, yielding only slight fluctuations.

*Metrol. Meas. Syst.*, Vol. XIX (2012), No. 2, pp. 177-190.

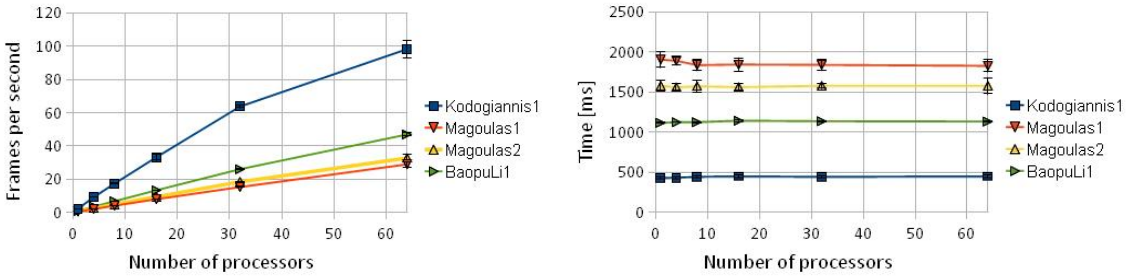


Fig. 7. Processing rate (left) and delay (right) of frame-level parallelized algorithms.

Each computational task in the Kaskada platform is actually a process that can be executed using multiple threads. Since Galera's nodes are 8-core systems, it is a reasonable choice to split the execution into 8 threads, which could potentially result in 8 times speedup in ideal case. In practice, however, the achieved speedup is usually much lower due to memory access conflicts and data synchronization between threads. The advantage of this solution is a possibility to shorten the single frame processing time, at the same time reducing the delay.

Multithreading was implemented using the OpenMP mechanism. To accomplish this it was required to identify time-consuming loops in the algorithms, which should be parallelized. Therefore, execution time measurements of particular stages of the algorithm were carried out, which indicated code regions to be parallized. Fig. 8 presents the achieved results according to the number of processors used for multithreading.

As expected, the method results in much lower speedups than the previous one. While algorithms BaopuLi1 and Kodogiannis1 gained satisfactory speedup with high efficiency, including shortenening the single frame time, algorithms Magoulas1 and Magoulas2 did not show the significant performance improvement. The reason for this fact is the low capability of these algorithms for parallelization implicated from large data dependencies. The method therefore enables a slight increase of the processing rate and the reduction of introduced delay. In the case of less complex algorithms this technique may be sufficient. It can be also succesfully pulled together with other methods like the previous concurrent frame processing or pipeline processing.

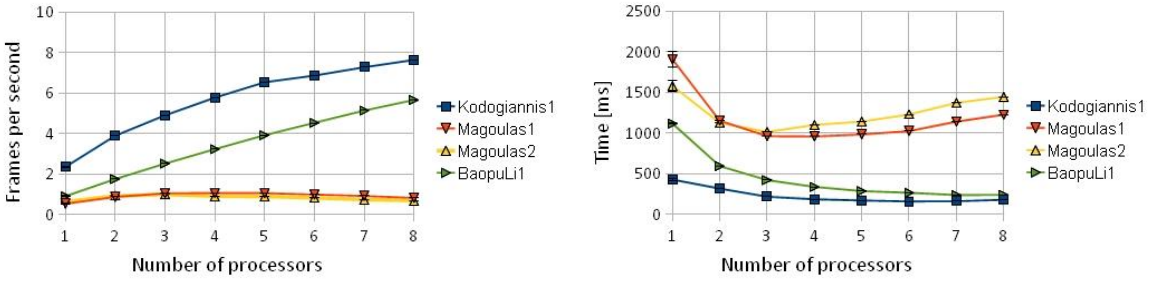


Fig. 8. Processing rate (left) and delay (right) of algorithms parallelized using multithreading.

The most interesting parallelization technique offered by the Kaskada platform is algorithm-level pipeline processing. The algorithms are divided into functional blocks to be executed by separate computational tasks in the form of a pipeline. Independent blocks can be put in a layer for concurrent execution. This allows to construct a service arranged adequately to a logic scheme of the algorithm. Therefore, each of the blocks can be distributed to different Galera's node and executed using multithreading, which enables to utilize a large number of processors. An exemplary service implementing such scenario for the algorithm BaopuLi1 is shown in Fig. 9.

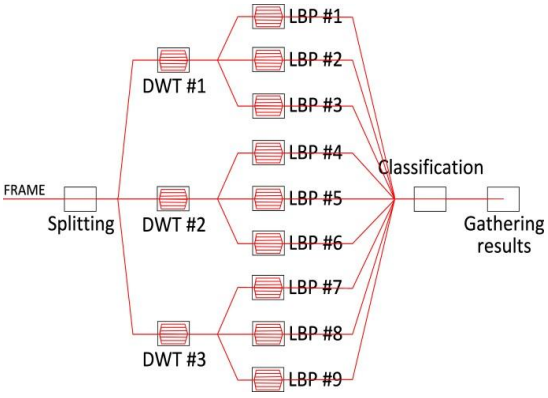


Fig. 9. Conception of pipeline processing combined with multithreading for BaopuLi1 algorithm.

Pipeline processing assures an increase of the processing rate, while concurrent execution of separate blocks, as well as multithreading, shortens the delay.

Fig. 10 shows the performance of algorithms measured for 8 variants utilising from 1 to 8 processors in each multithreaded node. Since the arrangement of the service is different for each of the algorithms, also the ranges of possibly used numbers of processors differ between them. Algorithm Magoulas1 was excluded from the test, since its structure prevents efficient pipeline implementation.

The BaopuLi1 algorithm showed relatively stable performance growth with the increasing number of processors. For 86 processors the processing rate exceeded 50, while the delay dropped below 0.1 s. The Kodogiannis1 algorithm achieved best performance for 32 processors. A marginal performance gain was achieved for the algorithm Magoulas2. The presented pipeline processing method therefore requires some sort of capability from the parallelized algorithm. In return, very good performance can be achieved for highly modular algorithms. This means that we can achieve a high level of dependability using an appropriate number of processors for analysis (e.g. for the Kodogiannis1 algorithm we must use at least 12 processors).

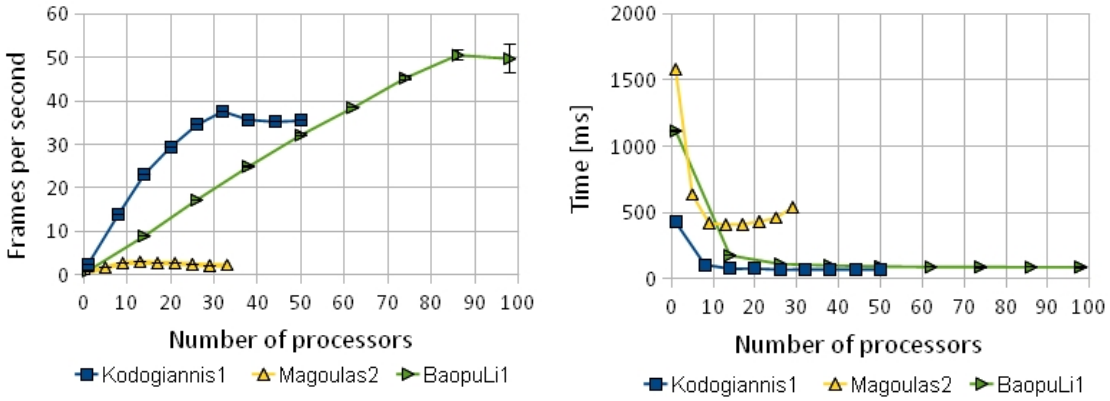


Fig. 10. Processing rate (left) and delay (right) of algorithms parallelized using pipeline model with multithreading.

For usability assessment we proposed to determine the average measure of the diagnostic time and the matching abnormal regions captured.

The results of the data analysis showed that the average viewing time was a 594 ± 93 s/sequence while the average scanning time was a 95 ± 25 s/sequence. The mean ratio of viewing time/scanning time was 6.25, which implies that the variations between the viewing and scanning time sequences were quite large. The average diagnostic time for each sequence, with the mean value being approximately 689 ± 118 s/sequence. The resulting average diagnostic time of the extracted sequence with a length of 307 minutes (18420 s) implies that

*Metrol. Meas. Syst.*, Vol. XIX (2012), No. 2, pp. 177-190.

when using the proposed method the time consumed is around 27% of the length of the sequence.

The total number of abnormal regions captured by doctor A was 74 regions. The numbers of abnormalities present differed with each sequence. For some sequences, there were from 2 to 6 abnormal regions, and thus the rate of matching in these sequences was high. For two sequences, however, it included 11 regions, and as it was the sequence with the maximum number of abnormalities present, it had a lower rate of matching. Overall, the average matching rate of the abnormal regions was 75% for one of the medical doctors, 70% for the second one. These results imply that although finding suspicious regions depends on other factors, such as one's personal judgment and skills, the concentration of the physicians as well as the number of abnormalities present on the video material. The proposed method produces acceptable rates of capture of relevant findings.

## 5. Conclusions

The paper presented two platforms BeesyCluster and Kaskada allowing rapid and easy composition of distributed applications out of ready-to-use services and components offering high usability to the user. These are especially useful for multimedia applications through the ability to quickly connect services and components, reuse already defined services and patterns, engage ready-to-use algorithms for optimization of QoS when running the application. For BeesyCluster, an application for parallel processing of digital images by distributed services was shown. It was demonstrated how granularity and either local or distributed environment influence the execution time and scalability and how to achieve desired QoS requirements involving execution time, reliability and cost. For Kaskada, an application for stream processing of endoscopic videos was presented. Parallelization capabilities of the Kaskada platform enabled a considerable performance gain for the investigated algorithms. The presented medical recognition algorithms suffered from high computational complexity, resulting in long execution time. Utilizing the computational power of the Galera supercomputer, Kaskada accelerated all the algorithms to perform fairly well in the offline processing mode, providing high speedups with an increasing number of processors. For sufficiently divisible algorithms, also online processing became possible by utilizing pipeline processing supported by multithreading.

Furthermore, the two platforms are complementary in terms of types of applications and QoS goals as indicated in Table 2.

Table 2. Preference of platforms for particular types of applications and QoS goals.

| BeesyCluster | Kaskada |
|---|---|
| 1. highly distributed workflows which integrate dedicated services offered by many providers from their own distributed servers/clusters.<br>2. workflows which require automatic multicriteria static or dynamic QoS service selection.<br>3. dependable distributed workflows offering a desired balance of performance, cost and reliability. | 1. multimedia workflows using highly specialized filters and services such as detecting specific objects on video streams or finding specific sounds in audio streams<br>2. performance oriented workflows thanks to dedicated infrastructure for communication between workflow nodes and middleware tuned for particular HPC clusters at Academic Computer Center, Gdansk University of Technology<br>3. real-time workflows based on services orchestration.<br>4. workflows related to public safety. |

## References

[1] Krafzig, D., Banke, K., Slama, D. (2004). *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall PTR.

[2] Krawczyk, H., Proficz, J. (2010). Kaskada – multimedia processing platform architecture. *In Proc. International Conference on Signal Processing and Multimedia Applications (SIGMAP 2010)*, 26-31.

[3] Czarnul, P. (2006). Reaching and Maintaining High Quality of Distributed J2EE Applications – BeesyCluster Case Study. Software Engineering Techniques SET 2006, Warsaw, Poland, in: *Software Engineering Techniques: Design for Quality*, ed. K. Sacha, Springer, International Federation for Information Processing, a Springer Series in Computer Science, 179-190.

[4] Krawczyk, H., Wiszniewski, B. (1998). *Analysis and testing of distributed software applications*. Baldock: Res. Stud. Press.

[5] Czarnul, P. (2010). Modeling, run-time optimization and execution of distributed workflow applications in the JEE-based BeesyCluster environment. *Journal of Supercomputing*. DOI: 10.1007/s11227-010-0499-7, Springer, 1-26.

[6] Czarnul, P., Matuszek, M., Wójcik, M., Zalewski, K. (2011). BeesyBees: A mobile agent-based middleware for a reliable and secure execution of service-based workflow applications in BeesyCluster. *Multiagent and Grid Systems Journal*, IOS Press, 7(6), 219-241.

[7] Czarnul, P. (2010). Modelling, Optimization and Execution of Workflow Applications with Data Distribution, Service Selection and Budget Constraints in BeesyCluster. In *Proceedings of 6th Workshop on Large Scale Computations on Grids and 1st Workshop on Scalable Computing in Distributed Systems*, International Multiconference on Computer, 5, 629-636.

[8] Wieczorek, M., Hoheisel, A., Prodan R. (2009). Towards a general model of the multi-criteria workflow scheduling on the grid. *Future Generation Computer Systems*, 25, 237-256.

[9] Yu, J., Buyya, R. (2005). A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3, 171-200.

[10] Yu, J., Buyya, R., Ramamohanarao, K. (2008). Workflow Scheduling Algorithms for Grid Computing. Springer. In *Metaheuristics for Scheduling in Distributed Computing Environments*, Berlin, Germany, 146, 173-214.

[11] Chin, S.H., Suh, T., Yu, H.C. (2010). Adaptive service scheduling for workflow applications in service-oriented grid. *Journal of Supercomputing*, 52, 253-283.

[12] Garg, S.K., Buyya, R., Siegel, J. (2010). Time and cost trade-off management for scheduling parallel applications on utility grids. *Future Generation Computer Systems*, 26, 1344-1355.

[13] *http://i.top500.org/system/9260* (January 2012).

[14] Blokus, A., Jedrzejewski, M., (2011). The design of an intelligent medical space supporting automated patient interviewing. In *Proceedings of the 5th International Conference of Young Scientists: Computer Science & Engineering*, Lviv, Ukraine, 16-19.

[15] Iakovidis, D.K., Tsevas, S., Polydorou, A. (2010). Reduction of capsule endoscopy reading times by unsupervised image mining. In *Computerized Medical Imaging and Graphics. Biomedical Image Technologies and Methods* - BIBE 2008, 34(6), 471-478.

[16] Li, B., Meng, M. (October 2009). Small bowel tumor detection for wireless capsule endoscopy images using textural features and support vector machine. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 498 - 503.

[17] Kodogiannis, V.S., Boulougoura, M. (2007). An adaptive neurofuzzy approach for the diagnosis in wireless capsule endoscopy imaging. *International Journal of Information Technology*, 13(1), 46-56.

[18] Magoulas, G.D., Plagianakos, V.P., Vrahatis, M.N. (2004). Neural network-based colonoscopic diagnosis using on-line learning and differential evolution. *Applied Soft Computing*, 4(4), 369-379.

[19] Magoulas, G.D. (2006). Neuronal networks and textural descriptors for automated tissue classification in endoscopy. *Oncology Reports*, 15, 997-1000.