Kempa-Liehr *Journal of Big Data* (2015) 2:10
DOI 10.1186/s40537-015-0017-0

Journal of Big Data
a SpringerOpen Journal

RESEARCH Open Access

Performance analysis of concurrent workflows

Andreas Kempa-Liehr[1,2,3]

Correspondence:
kempa-liehr@fmf.uni-freiburg.de
[1]Freiburg Materials Research Center, University of Freiburg, Stefan-Meier-Straße 21, 79104 Freiburg im Breisgau, Germany
[2]EnBW Energie Baden-Württemberg AG, Durlacher Allee 93, 76131 Karlsruhe, Germany
Full list of author information is available at the end of the article

Abstract

Automated workflows are the key concept of big data pipelines in science, engineering and enterprise applications. The performance analysis of automated workflows is an important topic of the continuous improvement process and the foundation of designing new workflows. This paper introduces the concept of process evolution functions and event reduction policies, which allow for the time resolved visualization of an unlimited number of concurrent workflows by means of aggregated task views. The visualization allows for an intuitive approach to the performance analysis of concurrent workflows. The theoretical foundation of this approach is applicable for workflows represented by directed acyclic graphs. It is explained on the basis of a simple IO-workflow model, which is typically found for distributed resource management systems utilized for many-task computing.

AMS subject classification: 68Mxx

Keywords: Big data pipeline; High performance computing; Complex event processing; Communicating sequential processes; Continuous improvement process

Introduction

Big data pipelines build on automated workflows which are executed in massively parallel applications. In order to improve the performance of automated workflows the respective systems and workflows have to be monitored [1, 2]. In general it has to be distinguished between the monitoring of continuously changing states of a distributed system, e.g. server load and bandwidth [3–5] and the monitoring of events [6, 7] indicating the progress of a specific workflow or even a group of concurrent workflows. In this context the classical monitoring approaches are combined with elements from complex event processing [8–10] in which the progress of a workflow is captured as an ordered set of events, the so-called trace or lifeline [11–13]. Applications comprise the identification of wait states in large scale simulations [14], the analysis of for-loops [15], or the visualization of execution and wait times in distributed systems [16].

Background and literature review

We observe that the performance analysis of workflows is either designed for a small number of workflows and delivers a detailed view on their timing and their dependencies applying the concept of task views, or it is considered on a very high level due to the shear number of concurrent tasks. In the latter case the available information is aggregated

to time series documenting the number of pending and finished tasks, which is crucial for the scalability of event based monitoring [17–20] and deriving scheduling strategies [21, 22].

In this paper we present a generic approach to the analysis of workflow performance by introducing aggregated task views, which are capable of analyzing both needs: Detailed information on the progress of concurrent workflows and scalability to a large, respectively very large number of concurrent workflows. While the application of aggregated task views to IO-workflows has been introduced in the course of a poster presentation at the International Supercomputing Conference 2010 [23], this paper discusses the theoretical foundation of aggregated task views on basis of process evolution functions and their policy-based aggregation.

However, the theoretical foundation of process evolution functions is not trivial, therefore we are following the ansatz of C. A. R. Hoare for introducing communicating sequential processes [24] and start with grossly oversimplified examples which are successively extended to more complex problems. Thus showing the universal applicability of analyzing the performance of automated workflows by means of aggregated task views. For this purpose we have chosen an example from the so-called capacity computing or many-task computing (MTC), which denotes high-performance computations comprising multiple distinct activities, coupled via file system operations or message passing [25]. These remote computations are often used for preparing high-performance computing (HPC) simulations considered as capability computing [26, 27]. MTC is often configured and run by means of a tool chain creating sets of parameterized computations, initiating the remote computations on a computing cluster and providing the results for further processing [28]. Therefore the IO-workflow of these tasks exhibit extensive pre- and post-processing phases which significantly influence the overall performance of the computational tasks. This bottleneck becomes even more important for data intensive computing [29, 30].

A central aspect of this paper is the utilization of temporal objects for purposes of performance analysis. Consequently, the nomenclature is in accordance with the glossaries of Jensen et al. [31] and Bettini et al. [32]. A comprehensive review on the treatment of time in computing is given in [33].

Research design and methodology

The remainder of this paper is organized as follows: The first section introduces aggregated task views on the basis of a sequential workflow. Then the concept of process evolution functions is derived, which is extended to MTC workflows. For scalability purposes some event reduction policies are introduced and the rendering of task views is explained. Finally, the example of a performance analysis is discussed. The paper closes with a review on related work and a conclusion.

Visualizing concurrent workflows

Consider a scientific or commercial application where a big data pipeline is configured on a workstation by collecting data from different information systems. Afterwards the data are transferred to a HPC cluster, where the data are analyzed by computing tasks, which are submitted to a Distributed Resource Management System (DRMS). After the computation of the tasks their result files are copied to the

initial workstation. In this example each computing task is processed by the following workflow:

1. transfer initial data from workstation to HPC cluster,
2. submit computing task to grid engine and wait for execution,
3. execute computing task,
4. transfer result data from HPC cluster to workstation.

The processing of a specific computing task by this workflow is called a job and a set of jobs is called job array. In our first example, there are no dependencies between the computing task such that all jobs can be started at once and parallelization can be used on appropriate infrastructure. However, each job progresses through five different states:

$$\text{job started} \rightarrow \text{task pending} \rightarrow \text{task running} \rightarrow \text{task finished} \rightarrow \text{job finished}.$$

In order to evaluate the performance of a MTC job array its progress can be visualized by means of task views. The general idea of task views is to draw a progress bar for each job while its drawing colour changes with respect to the state of the corresponding job. Progress bars from different jobs are stacked one below the other such that a two-dimensional graph is rendered.

In order to introduce the concept of aggregated task views, an ex-post analysis of 9253 data intensive computing tasks is shown in Fig. 1. The initial data of these jobs have been transferred from a workstation to a HPC cluster consisting of 10 Quad-Core computing nodes, which are managed by Univa Grid Engine [34]. The results of the computations have been transferred back to the original workstation. Of course it is not reasonable to draw 9253 individual progress bars in such a visualization, therefore the jobs have been clustered into 283 groups of approximately 32 jobs and for each job group an aggregated progress bar is drawn. In this example the *tasks pending* state of the aggregated progress bar starts with the first *task pending* event of any job of the respective job group and lasts until the first *task running* state of any job of the respective job group. Its *tasks running*

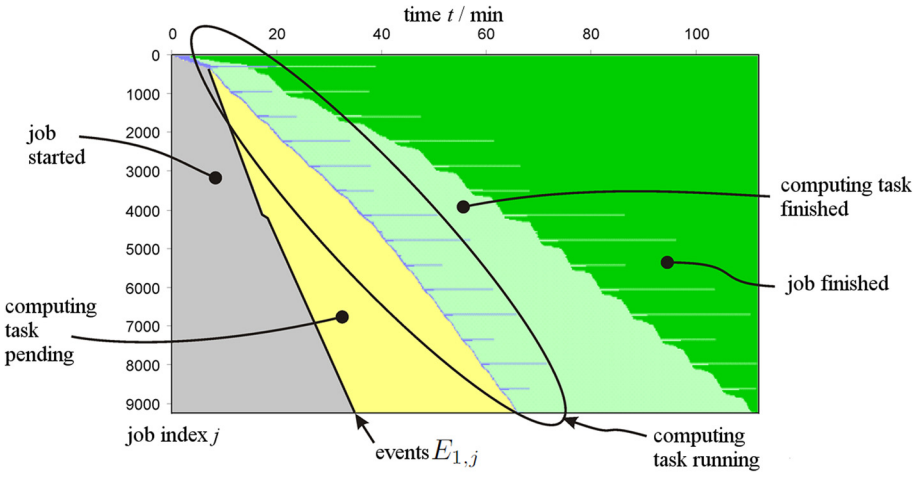

Fig. 1 Aggregated task view visualizing the time resolved progress of 9253 data intensive computing tasks, which have been transferred to a HPC cluster, where each task has been scheduled by Univa Grid Engine. Afterwards the results have been copied back to the original workstation. Coloured regions refer to jobs being in the same workflow state as time evolves from left to right. Slopes of interfaces between coloured regions indicate transfer rates between the respective workflow states

state continues until all jobs of the job group have finished computation, which is the starting point of the *tasks finished* state. The latter lasts until all jobs of the respective job group have signalled a *job finished* event.

The horizontal axis of the aggregated task view shown in Fig. 1 reveals that all 9253 jobs have been finished in approximately 110min. The coloured regions of the diagram indicate the five states of the workflow from which is visible that the majority of computing tasks execute less than one minute on the cluster system and only 14 groups contain at least one task, which computes up to 15 minutes on the cluster system. Note, that the incline of the interfaces between the coloured regions reveal the aggregated transfer rates between the different job states. Therefore, the first 360 jobs start their computation on the cluster system almost immediately after being submitted to the grid engine, which corresponds to an input data transfer rate from the workstation to the cluster system of approximately 50 jobs/minute. Afterwards the transfer rate increases to approximately 315 jobs/minute, which is accounted to the data transfer protocol dedublicating the input data. Because the HPC cluster finishes on average 150 jobs/minute a queue of pending tasks forms, which reaches a maximum of 5700 jobs approximately 35 minutes after the start of the job array. A more accurate look at the slope between the *tasks pending* state and the *tasks running* state reveals that after 40 minutes the slope increases from 130 tasks/minute to 170 jobs/minute, which reduces the computing time by 10 minutes. This is attributed to the fact, that cluster resources have been released from a job array running in parallel.

Process evolution functions

While the general concept of aggregated task views becomes quite clear from the example above, their theoretical foundation is not trivial. Therefore, as starting point, we are going to model the workflow of the MTC jobs visualized in Fig. 1 on basis of communicating sequential processes [24]. Here, the transition from one process state to the next state is triggered by a specific event E_i . In order to model our example the following events have been chosen:

E_0 : job starts to copy input data from workstation to HPC cluster,
E_1 : job submits computing task to batch queuing system,
E_2 : computing task is scheduled and starts execution,
E_3 : computing task is finished and job starts copying result data from HPC cluster to workstation,
E_4: job has finished copying the results data.

In terms of communicating sequential processes the modelled IO-workflow is a process

$$P = E_0 \rightarrow E_1 \rightarrow E_2 \rightarrow E_3 \rightarrow E_4 \rightarrow STOP_P \tag{1}$$

with alphabet $\alpha P = \{E_0, E_1, E_2, E_3, E_4\}$. The sequence

$$s = \langle E_0, E_1, E_2, E_3, E_4 \rangle \tag{2}$$

of events $E_0, \ldots, E_4$ represent the trace of process P [24] or so-called lifeline [11]. Recording the instant t of these events is a mapping

$$\tau : \alpha P \rightarrow \mathbb{T} \tag{3}$$

of the process alphabet αP to a time domain $\mathbb{T}$. This mapping realizes an ordered set of clocks $\{\tau(E_0), \ldots, \tau(E_4)\}$ with $\forall_i : \tau(E_i) < \tau(E_{i+1})$. The clocks are assumed to be synchronized even in a distributed system [35]. These clocks allow for the monitoring of the state $S_P(t|s)$ of process P with respect to its trace s as continuous function of time t by

$$S_P(t|s) = \sum_{E \in s} \theta\left(t - \tau(E)\right) = \sum_{i=0}^{4} \theta\left(t - \tau(E_i)\right). \tag{4}$$

Here, $\theta(\cdot)$ denotes the Heaviside step function [36]

$$\theta(x) = \begin{cases} 1, & x \geq 0, \\ 0, & x < 0. \end{cases} \tag{5}$$

In terms of our IO-workflow example P, the process evolution function $S_P(t|s)$ is bounded $0 \leq S_P(t|s) \leq 5$ and increases stepwise at the occurrence of an event $E \in s$ of trace s. Therefore the value of the process evolution function can be directly related to the state of process P as

$$S_P(t|s) = \begin{cases} 0, & t < E_0, \text{ job not started}, \\ 1, & E_0 \leq t < E_1, \text{ job started}, \\ 2, & E_1 \leq t < E_2, \text{ computing task pending}, \\ 3, & E_2 \leq t < E_3, \text{ computing task running}, \\ 4, & E_3 \leq t < E_4, \text{ computing task finished}, \\ 5, & t \geq E_4, \text{ job finished}. \end{cases} \tag{6}$$

An example of which is shown in Fig. 2. From a more general point of view the process evolution function $S_P(t|s)$ maps time t to process state index k for a given trace s. Because the IO-workflow example P does not exhibit cycles between process states, the relation between events $E \in s$ and process state index k simplifies to

$$S_P(t|s) = \begin{cases} k, & E_{k-1} \leq t < E_k, \\ 0, & \text{otherwise}. \end{cases} \tag{7}$$

Process evolution functions combine a specific process model P and the related event series of processes, which are inherently time-discrete, to a time-continuous analytical representation of the respective processes. It is important to note, that a process evolution function maps the recorded events of a process to a process state irrespectively of the

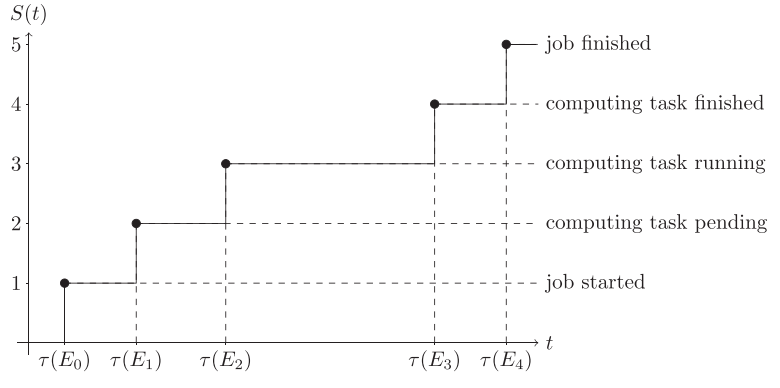

Fig. 2 Example of process evolution function $S(t|s)$ reflecting the progress of an IO-workflow with respect to its trace $s = \langle E_0, \ldots, E_4 \rangle$

number of recorded events. Therefore process evolution functions are useful for analyzing and optimizing live processes. Eg. in machine learning pipelines the process evolution function might be chosen as a robust feature.

Process evolution functions for MTC

For MTC we have to take account of the fact that in general a process or so-called job is defined by a directed acyclic graph (DAG). In terms of our previous example, this is exactly the case for a computing task submitting child tasks to the DRMS, in order to wait for the results and aggregate them later on (Fig. 3). Such child tasks have a reduced alphabet $\alpha\hat{P} = \{\hat{E}_1, \hat{E}_2, \hat{E}_3\}$ with process

$$\hat{P} = \hat{E}_1 \to \hat{E}_2 \to \hat{E}_3 \to STOP_{\hat{P}} \tag{8}$$

and events

$\hat{E}_1$: computing task submits child to batch queuing system,
$\hat{E}_2$: child task is scheduled and starts execution,
$\hat{E}_3$: child task is finished and provides results to parent task.

The DAG of a parent job, which submits three child jobs to a DRMS is shown in Fig. 4. In order to take account of the required scalability for monitoring MTC applications the discrimination between parent and child jobs has to be neglected. Therefore, child jobs inherit the *job started* event E_0 from its parent, such that their progress is documented by trace

$$\hat{s} = \langle E_0, \hat{E}_1, \hat{E}_2, \hat{E}_3 \rangle \tag{9}$$

and process evolution function

$$S_{\hat{P}}(t|\hat{s}) = \sum_{E \in \hat{s}} \theta\left(t - \tau(E)\right). \tag{10}$$

With this definition at hand it becomes clear, that process evolution functions $S_P(t|s)$ given in Eq. (4) and $\hat{S}_{\hat{P}}(t|\hat{s})$ map to identical process state indices (6). The only difference is that child jobs do not throw a *job finished* event from which follows $0 \leq S_{\hat{P}}(t|\hat{s}) \leq 4$. However, this is acceptable, because the results of child tasks are aggregated by the parent task, which is responsible for delivering the *job finished* event (Fig. 4).

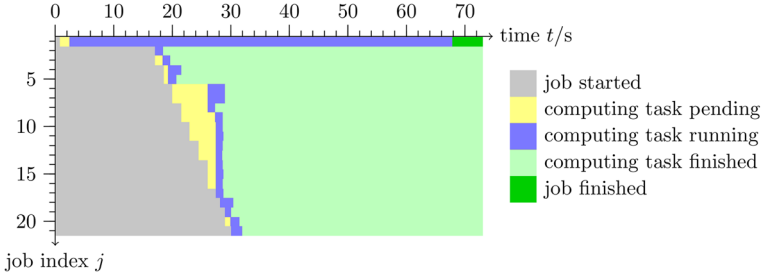

Fig. 3 Task view of parent job submitting twenty child jobs to Univa grid engine. Top row represents process evolution function of parent job. Subordinate rows correspond to child jobs. Note, that some jobs are immediately assigned to a worker nodes, such that no *Pending* state is visible. The computing task of the parent job finishes within 68 seconds. In this example the parent job uses the results of the child jobs, which are finished after 32 seconds, for running another optimization computation, which is not parallelized

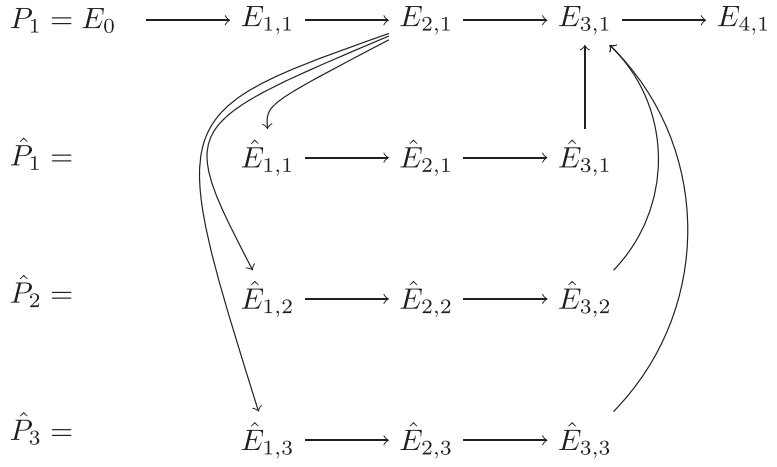

Fig. 4 Example of parent job P_1 from which three child jobs $\hat{P}_1$, $\hat{P}_2$, and $\hat{P}_3$ are submitted to batch queuing system

With this simplification at hand, parent and child jobs of MTC applications are regarded as indistinguishable jobs of job array $\{P_j\}$ with processes

$$P_j = E_0 \to E_{1,j} \to E_{2,j} \to E_{3,j} \to E_{4,j} \to STOP_{P_j} \tag{11}$$

for $j = 1, \ldots, N$ and alphabet $\alpha P_j = \{E_0, E_{1,j}, E_{2,j}, E_{3,j}, E_{4,j}\}$. Here, it is assumed that all jobs of job array $\{P_j\}$ are started by the same event E_0. The traces $\{s_j\}$ of job array $\{P_j\}$ are given by

$$s_j = \langle E_0, E_{1,j}, E_{3,j}, E_{3,j}, E_{4,j} \rangle \tag{12}$$

and its N-dimensional process evolution function

$$\begin{aligned} \vec{S}_P(t|\{s_j\}) &= \vec{S}_P(t|s_1, \ldots, s_j, \ldots, s_N) \\ &= \big(S_P(t|s_1), \ldots, S_P(t|s_j), \ldots, S_P(t|s_N)\big)^{\mathrm{T}} \end{aligned} \tag{13a}$$

with

$$S_P(t|s_j) = \sum_{E \in s_j} \theta\left(t - \tau(E)\right), \tag{13b}$$

an example of which is visualized in Fig. 3 as task view. Each element $S_P(t|s_j)$ represents an individual process evolution function, which can be independently plotted (e.g. Fig. 2).

The chosen representation $\vec{S}_P(t|\{s_j\})$ is convenient because it allows to monitor e.g. the amount $p_k(t)$ of jobs being in a specific state k at time t by simply counting the number of process evolution functions holding $S_P(t|s_j) = k$ and dividing this number by the number N of jobs:

$$p_k(t) = \frac{1}{N} \sum_{j:S_P(t|s_j)=k} 1. \tag{14}$$

A snapshot of all job states $\{p_0(t'), p_1(t'), \ldots, p_K(t')\}$ at instant $t = t'$ is often summarized as pie chart. They are often found as counting values in a workflow log file with timestamps.

Event reduction policies

In order to visualize the N-dimensional process evolution function $\vec{S}_P(t|\{s_j\})$ for large job arrays (e.g. Fig. 1), the information density has to be reduced whereby balancing the completeness of the picture and the significance of certain events. Therefore, in view of several thousands jobs to be analyzed, the dimension of process evolution function $\vec{S}_P(t|\{s_j\})$ has to be reduced.

We begin by aggregating the N-dimensional vector function $\vec{S}_P(t|\{s_j\})$ of job array $\{P_j\}$ to a scalar function $S_P(t|\tilde{s})$ by applying event reduction policies. These policies aggregate the N traces $\{s_j\}$ of job array $\{P_j\}$ to a single trace $\tilde{s}$ containing the most important events of the job array. The central idea is to sort all events $E_{k,j} \in \{s_j\}$, which initiate a certain process state k, with respect to their occurrence $\tau\left(E_{k,j}\right)$ and choose, depending on the reduction policy, the first or the last event of the considered process state k.

Therefore the reduction policies provide a filter for emphasizing certain process states. E.g. in the IO-workflow example it might be important for the user to identify the time interval in which at least one job is in state *task running*. Therefore we choose the reduction policies by selecting the first *task pending* event

$$\tilde{E}_1 = \underset{E:E_1\in\{s_j\}}{\arg\min}\ \tau(E)\,, \tag{15a}$$

the first *task running* event

$$\tilde{E}_2 = \underset{E:E_2\in\{s_j\}}{\arg\min}\ \tau(E)\,, \tag{15b}$$

the last *task finished* event

$$\tilde{E}_3 = \underset{E:E_3\in\{s_j\}}{\arg\max}\ \tau(E)\,, \tag{15c}$$

and the last *job finished* event

$$\tilde{E}_4 = \underset{E:E_4\in\{s_j\}}{\arg\max}\ \tau(E)\,. \tag{15d}$$

Consequently, the time intervall in which at least one job is in state *task running* starts at $\tau\left(\tilde{E}_2\right)$ and ends at $\tau\left(\tilde{E}_3\right)$. The reduced trace $\tilde{s}$ is given by

$$\tilde{s} = \langle E_0, \tilde{E}_1, \tilde{E}_2, \tilde{E}_3, \tilde{E}_4\rangle \tag{16}$$

and its aggregated process evolution function $S_P(t|\tilde{s})$ computes from Eq. (4). It maps time t to the aggregated job array state (Table 1) indicating that the job array $\{P_j\}$ has not been started $S_P(t|\tilde{s}) = 0$, input data are copied to the HPC cluster $S_P(t|\tilde{s}) = 1$, at least one computing task has been submitted $S_P(t|\tilde{s}) = 2$, at least one computing task is executing $S_P(t|\tilde{s}) = 3$, all computing task have been finished $S_P(t|\tilde{s}) = 4$, and all jobs have been finished $S_P(t|\tilde{s}) = 5$.

Now consider that too much information might be discarded by this aggregation and a more detailed view on the progress of job array $\{P_j\}$ is needed. Therefore, the event

Table 1 Interpretation of aggregated process evolution function $S_P(t|\tilde{s})$

Interval	$S_P(t\|\tilde{s})$	Interpretation
$t < \tau(E_0)$	0	Job array $\{P_j\}$ has not been started
$\tau(E_0) \leq t < \tau(\tilde{E}_1)$	1	Input data are being copied to the HPC cluster
$\tau(\tilde{E}_1) \leq t < \tau(\tilde{E}_2)$	2	At least one computing task has been submitted
$\tau(\tilde{E}_2) \leq t < \tau(\tilde{E}_3)$	3	At least one computing task is executing
$\tau(\tilde{E}_3) \leq t < \tau(\tilde{E}_4)$	4	All computing task have been finished
$\tau(\tilde{E}_4) \leq t$	5	All jobs have been finished

Events E_i are registered by a set of synchronized clocks $\tau(E_i)$

reduction is not applied to all traces $\{s_j\}$. Instead, the job array is split into M job groups of approximately M/N jobs with the exact number of jobs per group being retrieved from a variant of Bresenham's line algorithm [37]. Finally, the event reduction policies are applied to each of the job groups respectively their set of traces resulting into M reduced traces $\tilde{s}_1, \ldots, \tilde{s}_M$, which can be visualized with respect to their process evolution function $S_P(t|\tilde{s}_1), \ldots, S_P(t|\tilde{s}_M)$ by means of an aggregated process evolution function

$$\vec{\tilde{S}}_p(t|\tilde{s}_1, \ldots, \tilde{s}_M) = (S_P(t|\tilde{s}_1), \ldots, S_P(t|\tilde{s}_M))^{\mathrm{T}}, \tag{17}$$

which is an M-dimensional vector function being the analytical representation of an aggregated task view, an example of which is shown in Fig.1.

Results and discussion

Rendering task views

In the previous sections it has been shown that process evolution functions are an analytical time continuous representation of event series. Therefore they are useful for rendering task views because they allow for the transformation of event series into discrete time signals by sampling the state $S_P(t|s)$ of process P.

In this paper task views have been rendered as raster graphics of $Y \times X$ pixels with Y denoting the number of rows and X denoting the number of columns. In the examples given in Figs. 1, 3, and 5 time is the abscissa such that the following relation between the rendered time intervall $t \in [t_0, t_0 + T]$, the number of columns X, and the inverse sampling rate Δ_t, holds

$$\Delta_t = \frac{T}{X - 1}. \tag{18}$$

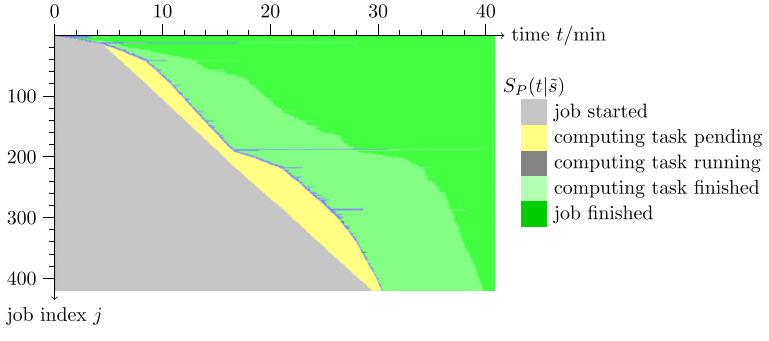

Fig. 5 Aggregated task view of 419 MTC jobs processed by a HPC cluster. The job throughput of this example could be increased by rearranging the jobs with respect to their computing time, and compressing results files in order to reduce the time taken for copying them

With this relation at hand a sample $S_{P,x}$ of process evolution function $S_P(t|s)$ is given as

$$S_{P,x} = S_P\left(t_0 + (x-1)\Delta_t|s\right) \text{ with } x = 1, 2, \ldots, X. \tag{19}$$

Consequently, the discrete time signal $\vec{S}_P$ of process P is a process evolution sequence

$$\begin{aligned}\vec{S}_P &= \left(S_{P,1}, S_{P,2}, \ldots, S_{P,x}, \ldots S_{P,X}\right) \\ &= \left(S_P(t_0|s), S_P(t_0+\Delta_t|s), \ldots, S_P(t_0+(x-1)\Delta_t|s), \ldots, S_P(t_0+T|s)\right). \end{aligned} \tag{20}$$

If the number of rows Y is equal to the number of processes N exactly one sequence $\vec{S}_P$ quantifies the pixel values of one row of the raster graphics. If the number of rows Y is larger than the number of processes N the event reduction policies (15) are applied for computing $M = Y$ reduced traces $\tilde{s}_1, \ldots, \tilde{s}_M$, from which the process evolution sequences (20) are computed (Fig. 1). The last scenario comprises situations, where the number of rows Y is smaller than the number N of processes. In this case the Y rows are split into N groups of approximately Y/N rows per job. Consequently, each of the N process evolution sequences is repeatedly used for filling the pixels of its corresponding row group. The exact distribution of rows on process evolution sequences might be retrieved from Bresenham's line algorithm [37]. An example of this kind of task view is shown in Fig. 3.

Analyzing job array performance

Frequently, the question arises whether the throughput of an MTC application can be increased by increasing the number of computing nodes or by refactoring the underlying algorithms with respect to the MTC workflow. In order to decide on this question an aggregated task view of a representative job array might provide useful information.

In order to demonstrate such kind of analysis an aggregated task view visualizing the performance of 419 jobs is shown in Fig. 5. Similar to the introductory example the jobs have been configured to transfer the input data to a HPC cluster, submit the computing task to Univa Grid Engine, wait for their execution and copy the results back to the original workstation. The aggregated task view reveals that the job array contains at least two computing tasks running up to 15 minutes, while the majority of computing tasks has finished in less than a minute. It becomes clear that one of the long running jobs finishes after all other jobs have been finished. Therefore, rearranging the job array such that jobs with long execution times are started right at the beginning will be a first measure of increasing the job array performance.

The aggregated task view also shows that the last ten minutes of processing the job array is spent for copying result files from the HPC cluster to the original workstation. There is a good chance, that compressing the result files will decrease the duration of the final file transfer significantly.

The rate of jobs switching from *Pending* to *Running* state changes frequently without significant changes of the runtime of the computing tasks, which indicates that the HPC cluster is shared with job arrays running in parallel. Most of the time the queue of *Pending* states is filled as fast as computing tasks finishing execution such that only a small stack of pending tasks forms. From this observation follows that the job throughput would only improve from increasing the number of computing nodes if additionally the time taken for transferring the input files to the HPC cluster could be decreased.

However, although such kind of performance issues could be easily detected by established monitoring tools, the example demonstrates that aggregated task views are a useful tool not only for system administrators but also for users of MTC.

Related work

Monitoring of concurrent workflows is an important issue in computational science and information technology due to its relevance for running complex IT-systems and providing stable IT-services. Therefore, nearly every workflow management system includes a monitoring component. On basis of a selection of established workflow management and monitoring frameworks we are discussing the integration ability of aggregated task views.

P-GRADE is a portal for managing grid computing workflows on a variety of grid systems [38]. It includes the monitoring program Prove [39], which is able to visualize task views from trace files of individual jobs. Due to its combination of an existing workflow model and the capturing of traces the concept of aggregated task views could be easily integrated into P-GRADE either by chosen default aggregation policies are by providing a configuration interface for aggregation policies.

ASKALON is a cloud and grid application development and computing environment, which is designed as a distributed service-oriented architecture [40]. Two services are important to mention in this context: the performance prediction service and the performance analysis service. Both services benefit from a rich set of performance metrics including metrics operating on workflow events. Combining the recorded events with the specified workflow model and an additional configuration layer for aggregation policies, the presented concept of aggregated task views could be integrated into ASKALON. Furthermore, the concept of process evolution functions might even extend the existing performance analysis capabilites of ASKALON, because it maps time-discrete event series to a time-continuous representation, which calls for an integrated analysis of time-discrete and time-continous performance attributes.

Web Service Navigator is a visualization tool for service-oriented architecture applications, which among others is used for analysing performance bottle necks [41]. It correlates metric information about web services requests and responses, models the transactions they represent, and extracts a process model, which is visualized as task view and is used to access performance statistics on selected transactions. Due to its automatic conduction of a process model and the collection of transaction events the Web Service Navigator could be enhanced by rendering aggregated task views of concurrent transactions in order to simplify the identification of performance bootle necks. Again, these aggregated task views might be configured by an additional configuration layer for aggregation policies.

Of course, there are also theoretical approaches for analyzing the performance of workflows like e.g. the quality of service model introduced by Cardoso et al. [42] and the performance ontologies of Truong et al. [43]. The theoretical foundation of these papers clearly inherits from the algebra of communicating sequential processes introduced by Hoare [24], which is also the starting point of this paper. However, both approaches [42, 43] estimate performance attributes by averaging over the history of all known performance attributes, such that short-scale trends in performance fluctuations are difficult to identify. Consequently, these approaches cannot deal with distributions of performance attributes exhibiting systematic outliers, like the ones depicted in Fig. 1. This is a clear

advantage of the approach presented in this paper, because it provides an intuitive performance evaluation, which is extremly useful for the users of workflow management systems.

Conclusions

The task views discussed in this paper are snapshots, which have been taken from the implementation of a productive workflow management system running at EnBW Energie Baden Württemberg AG [23]. This system is used for running time-critical data-intensive jobs conducting operational research computations for asset planning purposses and energy market analysis.

We have introduced a new paradigm for the performance analysis of MTC workflows on basis of process evolution functions and their policy based aggregation leading to the visualization of aggregated task views. The concept has been explained on basis of IO-workflow monitoring with the computational tasks being scheduled by Univa Grid Engine to a HPC cluster.

The presented analysis paradigm introduces process evolution functions in order to solve the problem of analyzing event series, which are inherently time-discrete, within the context of concurrent workflows by mapping them to a time-continuous representation. The results of this approach are promising with respect to its detailed view on workflow progress and its scalability to thousands of concurrent jobs. It can be directly applied to workflows described by DAGs especially in highly automated systems like big data pipelines, the P-GRADE grid portal [38], ASKALON [40], Petascale science applications [44], the development of new HPC services like hybrid computing [26], or even event-driven business processes [45, 46] and the visualization of web services [41].

We expect that the visualization paradigm of aggregated task views will be integrated into established monitoring tools in order to provide a scalable time-resolved visual performance analysis of concurrent workflows. Furthermore, the concept of process evolution functions might become important in data science and big data applications in the context of process optimization projects, because it provides a robust feature for characterizing the state of running workflows.

Competing interests

The concept of process evolution functions and the first draft of this paper have been developed in the context of the CIO agenda topic *High Performance Computing* at the department of corporate IT strategy at EnBW Energie Baden-Württemberg AG. The presented theoretical framework and the final manuscript have been written in the course of the associated membership of the author at the Freiburg Materials Research Center.

Acknowledgements

The author likes to thank Energy Solution Center e.V. for its networking support, A. Treml for the opportunity to develop the theoretical foundation of this paper, A. Celan for fruitful discussions in an early stage of the manuscript, and Th. Fanslau for the original visualization concept. The article processing charge was funded by the German Research Foundation (DFG) and the Albert Ludwigs University Freiburg in the funding programme Open Access Publishing.

Author details

[1]Freiburg Materials Research Center, University of Freiburg, Stefan-Meier-Straße 21, 79104 Freiburg im Breisgau, Germany. [2]EnBW Energie Baden-Württemberg AG, Durlacher Allee 93, 76131 Karlsruhe, Germany. [3]Blue Yonder GmbH, Ohiostraße 8, 76149 Karlsruhe, Germany.

Received: 9 January 2015 Accepted: 19 May 2015
Published online: 03 July 2015

References

1. Iosup A, Ostermann S, Yigitbasi MN, Prodan R, Fahringer T, Epema DHJ (2011) Performance analysis of cloud computing services for many-tasks scientific computing. Parallel Distributed Syst IEEE Trans 22(6):931–945. doi:10.1109/TPDS.2011.66

2. Zhan J, Zhang L, Sun N, Wang L, Jia Z, Luo C (2012) High volume throughput computing: Identifying and characterizing throughput oriented workloads in data centers. In: Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International. IEEE, Piscataway (NJ). pp 1712–1721. doi:10.1109/IPDPSW.2012.213
3. Hollingsworth J, Tierney B (2003) Instrumentation and Monitoring. In: Foster I, Kesselmann C (eds). The Grid 2: Blueprint for a New Computing Infrastructure. The Elsevier Series in Grid Computing. Elsevier Science, Amsterdam. pp 319–352
4. Zanikolas S, Sakellariou R (2005) A taxonomy of grid monitoring systems. Futur Gener Comput Syst 21:163–188. doi:10.1016.j.future.2004.07.002
5. Kokkinos P, Varvarigos EA (2012) Scheduling efficiency of resource information aggregation in grid networks. Futur Gener Comput Syst 28(1):9–23. doi:10.1016/j.future.2011.06.008
6. Brunner P, Truong HL, Fahringer T (2006) Performance monitoring and visualization of grid scientific workflows in ASKALON. In: Gerndt M, Kranzlmüller D (eds). High Performance Computing and Communications. Lecture Notes in Computer Science. Springer, Berlin Heidelberg Vol. 4208. pp 170–179. doi:10.1007/11847366_18
7. Wang TD, Plaisant C, Shneiderman B, Spring N, Roseman D, Marchand G, Mukherjee V, Smith M (2009) Temporal summaries: Supporting temporal categorical searching, aggregation and comparison. Vis Comput Graph IEEE Trans 15(6):1049–1056. doi:10.1109/TVCG.2009.187
8. Suntinger M, Schiefer J, Obweger H, Groller ME (2008) The event tunnel: Interactive visualization of complex event streams for business process pattern analysis. In: Visualization Symposium, 2008. PacificVIS '08. IEEE, Piscataway (NJ). pp 111–118. doi:10.1109/PACIFICVIS.2008.4475466
9. Moser O, Rosenberg F, Dustdar S (2010) Event driven monitoring for service composition infrastructures. In: Chen L, Triantafillou P, Suel T (eds). Web Information Systems Engineering - WISE 2010. Lecture Notes in Computer Science. Springer, Berlin Heidelberg Vol. 6488. pp 38–51. doi:10.1007/978-3-642-17616-6_6
10. Balis B, Kowalewski B, Bubak M (2011) Real-time grid monitoring based on complex event processing. Futur Gener Comput Syst 27:1103–1112. doi:10.1016/j-future.2011.04.005
11. Tierney B, Johnston W, Cowley B, Hoo G, Brooks C, Gunter D (1998) The netlogger methodology for high performance distributed systems performance analysis. In: In Proc. 7th IEEE Symp. on High Performance Distributed Computing. IEEE, Piscataway (NJ). pp 260–267
12. Gunter DK, Jackson KR, Konerding DE, Lee J, Tierney B (2005) Essential grid workflow monitoring elements. In: Arabnia HR, Ni J (eds). GCA. CSREA Press, Athens. pp 39–45. http://acs.lbl.gov/publications/NetLogger-GID.pdf
13. Bailey SJ, Tierney B, Gunter DK (2005) Scalable analysis of distributed workflow traces. In: Arabnia HR (ed). PDPTA. CSREA Press, Athens. pp 849–855
14. Böhme D, Geimer M, Wolf F, Arnold L (2010) Identifying the root causes of wait states in large-scale parallel applications. In: Proc. of the 39th International Conference on Parallel Processing (ICPP), San Diego, CA, USA. IEEE Computer Society, Washington, DC. pp 90–100. doi:10.1109/ICPP.2010.18. Best Paper Award
15. Nou R, Juliá F, Hogan K, Torras J (2011) A path to achieving a self-managed grid middleware. Futur Gener Comput Syst 27:10–19. doi:10.1016/j.juture.2010.07.002
16. Zhao Y, Raicu I, Foster I, Hategan M, Nefedova V, Wilde M (2008) Realizing Fast, Scalable and Reliable Scientific Computations in Grid Environments. In: Wong J (ed). Grid Computing Research Progress. Nova Science Publishers, Hauppauge, NY. pp 1–40
17. Kunz T, Black JP (1995) Using automatic process clustering for design recovery and distributed debugging. Softw Eng IEEE Trans 21(6):515–527. doi:10.1109/32.391378
18. Kunz T (1997) High-level views of distributed executions: convex abstract events. Automated Softw Engg. 4(2):179–197. doi:10.1023/A:1008685117587
19. Kunz T, Khouzam M (2000) Concurrent single stepping in event-visualization tools. Cluster Comput 3(3):231–243. doi:10.1023/A:1019092506798
20. Kranzlmüller D (2000) Event graph analysis for debugging massively parallel programs. PhD thesis,. Johannes Kepler University, Linz
21. Mandal A, Kennedy K, Koelbel C, Marin G, Mellor-Crummey J, Liu B, Johnsson L (2005) Scheduling strategies for mapping application workflows onto the grid. In: High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium On. IEEE, Piscataway (NJ). pp 125–134. doi:10.1109/HPDC.2005.1520947
22. Zhang Y, Mandal A, Koelbel C, Cooper K (2009) Combined fault tolerance and scheduling techniques for workflow applications on computational grids. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. CCGRID '09. IEEE Computer Society, Washington, DC, USA. pp 244–251. doi:10.1109/CCGRID.2009.59
23. Liehr AW, Fanslau T, Celan A (2010) Monitoring IO-Workflow by Means of State-Time Diagrams. In: International Supercomputing Conference (ISC). ResearchGate, Berlin. doi:10.13140/2.1.4818.6244
24. Hoare CAR (1985) Communicating Sequential Processes. June 21th 2004 edn. Prentice Hall International, Upper Saddle River, NJ. http://www.usingcsp.com/cspbook.pdf
25. Raicu I, Foster IT, Zhao Y (2008) Many-task computing for grids and supercomputers. In: Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008. Workshop On. IEEE, Piscataway (NJ). pp 1–11. doi:10.1109/MTAGS.2008.4777912
26. Mateescu G, Gentzsch W, Ribbens CJ (2011) Hybrid computing where HPC meets grid and cloud computing. Futur Gener Comput Syst 27(5):440–453. doi:10.1016/j.future.2010.11.003
27. Liehr AW, Moskalenko A, Bode M, Purwins HG (2000) Intergradient Simulations of Dissipative Quasi-particle Interactions with Solutions of a Three-component Three-dimensional Reaction-diffusion System. In: 6th Granada Seminar on Computational Physics. ResearchGate, Berlin. doi:10.13140/RG.2.1.5131.6641
28. Deelman E, Gannon D, Shields M, Taylor I (2009) Workflows and e-science: An overview of workflow system features and capabilities. Futur Gener Comput Syst 25(5):528–540. doi:10.1016/j.future.2008.06.012
29. Furht B, Escalante A (eds) (2011) Handbook of Data Intensive Computing. Springer, Berlin Heidelberg

30. Kosar T (ed) (2012) Data Intensive Distributed Computing: Challenges and Solutions for Large-Scale Information Management. Information Science Reference, Hershey, PA
31. Jensen C, Dyreson C, Böhlen M, Clifford J, Elmasri R, Gadia S, Grandi F, Hayes P, Jajodia S, Käfer W, Kline N, Lorentzos N, Mitsopoulos Y, Montanari A, Nonen D, Peressi E, Pernici B, Roddick J, Sarda N, Scalas M, Segev A, Snodgrass R, Soo M, Tansel A, Tiberio P, Wiederhold G (1998) The consensus glossary of temporal database concepts – February 1998 version. In: Etzion O, Jajodia S, Sripada S (eds). Temporal Databases: Research and Practice. Lecture Notes in Computer Science. Springer, Berlin Heidelberg Vol. 1399. pp 367–405. doi:10.1007/BFb0053710
32. Bettini C, Dyreson CE, Evans WS, Snodgras RT, Wang XS (1998) A glossary of time granularity concepts. In: Etzion O, Jajodia S, Sripada S (eds). Temporal Databases: Research and Practice. Lecture Notes in Computer Science. Springer, Berlin Heidelberg Vol. 1399. pp 406–413. doi:10.1007/BFb0053711
33. Furia CA, Mandrioli D, Morzenti A, Rossi M (2010) Modeling time in computing: A taxonomy and a comparative survey. ACM Comput Surv 42(2):6–1659. doi:10.1145/1667062.1667063
34. Univa Corporation (2014) Univa grid engine. Technical report, Univa Corporation. http://www.univa.com/resources/files/gridengine.pdf
35. Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7):558–565. doi:10.1145/359545.359563
36. Kanwal RP (1998) Generalized Functions: Theory and Technique. 2nd edn. Birkhäuser, Boston, MA
37. Bresenham JE (1965) Algorithm for computer control of a digital plotter. IBM Syst J 4(1):25–30. doi:10.1147/sj.41.0025
38. Kacsuk P, Kiss T, Sipos G (2008) Solving the grid interoperability problem by P-GRADE portal at workflow level. Futur Gener Comput Syst 24(7):744–751. doi:10.1016/j.future.2008.02.008
39. P-GRADE Portal Developer Alliance (2010) The prove program. P-GRADE Portal 2.10, MTA SZTAKI LPDS. http://portal.p-grade.hu/manual/user/v210/UsersManualRelease2_10.html#3_The_Prove_program_
40. Qin J, Fahringer T (2012) Scientific Workflows: Programming, Optimization, and Synthesis with ASKALON and AWDL. Springer, Berlin
41. Pauw W, Lei M, Pring E, Villard L, Arnold M, Morar J (2005) Web service navigator: Visualizing the execution of web services. IBM Syst J 44(4):821–845
42. Cardoso J, Sheth AP, Miller JA, Arnold J, Kochut KJ (2004) Quality of service for workflows and web service processes. Web Semantics: Sci Serv Agents World Wide Web 1(3):281–308
43. Truong HL, Dustdar S, Fahringer T (2007) Performance metrics and ontologies for grid workflows. Futur Gener Comput Syst 23(6):760–772. doi:10.1016/j.future.2007.01.003
44. Baranovski A, Bharathi S, Bresnahan J, Chervenak A, Foster I, Fraser D, Freeman T, Gunter D, Jackson K, Keahey K, Kesselman C, Konerding DE, Leroy N, Link M, Livny M, Miller N, Miller R, Oleynik G, Pearlman L, Schopf JM, Schuler R, Tierney B (2007) Enabling distributed petascale science. J Phys: Conference Series 78(1):012020. doi:10.1088/1742-6596/78/1/012020
45. Durand J, Cho H, Moberg D, Woo J (2011) XTemp: Event-driven testing and monitoring of business processes: Leveraging XML, XPath and XSLT for a practical event processing. In: Proceedings of Balisage: The Markup Conference 2011. Balisage Series on Markup Technologies. Mulberry Technologies, Inc., Rockville (MD) Vol. 7. doi:10.4242/BalisageVol7.Durand01
46. Luckham D (2012) Event Processing for Business. Organizing the Real-Time Enterprise. John Wiley & Sons, Hoboken, New Jersey