Journal of Integrative Bioinformatics, 8(2):163, 2011 http://journal.imbio.de

Data partitioning enables the use of standard SOAP Web Services in genome-scale workflows.

Paweł Sztromwasser[1,2*], Pål Puntervoll[2], Kjell Petersen[2]

[1]Department of Informatics, University of Bergen, `http://www.uib.no/ii`

[2]Computational Biology Unit, Uni Computing, Uni Research, Norway
`http://www.computing.uni.no/`

Summary

Biological databases and computational biology tools are provided by research groups around the world, and made accessible on the Web. Combining these resources is a common practice in bioinformatics, but integration of heterogeneous and often distributed tools and datasets can be challenging. To date, this challenge has been commonly addressed in a pragmatic way, by tedious and error-prone scripting. Recently however a more reliable technique has been identified and proposed as the platform that would tie together bioinformatics resources, namely Web Services. In the last decade the Web Services have spread wide in bioinformatics, and earned the title of recommended technology. However, in the era of high-throughput experimentation, a major concern regarding Web Services is their ability to handle large-scale data traffic. We propose a stream-like communication pattern for standard SOAP Web Services, that enables efficient flow of large data traffic between a workflow orchestrator and Web Services. We evaluated the data-partitioning strategy by comparing it with typical communication patterns on an example pipeline for genomic sequence annotation. The results show that data-partitioning lowers resource demands of services and increases their throughput, which in consequence allows to execute in-silico experiments on genome-scale, using standard SOAP Web Services and workflows. As a proof-of-principle we annotated an RNA-seq dataset using a plain BPEL workflow engine.

1 Introduction

Combining scientific resources is vital for acquiring a complete picture of a scientific problem, and plays a key role in the process of generating new knowledge. A plethora of tools and databases hosted at different sites around the globe are available to the life sciences community, and constitute a vast mine of information. Integration of all these distributed resources creates new perspectives, enables scientists to ask broader questions, but it also poses new challenges. Cooperation between distributed heterogeneous software systems is one such key challenge.

The challange is often addressed by ad-hoc scripting, that tightly couples required resources. This pragmatic approach is a tedious and error-prone process, so recently a more promising method has gained significant attention. SOAP Web Services have been proposed as the technology that can connect the distributed, heterogeneous bioinformatics resources [1]. They have

[*]To whom correspondence should be addressed: pawel.sztromwasser@ii.uib.no

been chosen for several reasons: SOAP Web Services are a W3C [1] standard for interoperable communication over the Internet; they provide mechanisms for semantic annotation, and automatic discovery and invocation; it is a well-established technology with large user and developer communities being a guarantee of sustainable support; they come along with good documentation, tools and libraries. Furthermore, in accordance with Service Oriented Architecture principles where simple components are loosely orchestrated to provide higher-order functionality, SOAP Web Services can be combined into workflows. In a scientific context, these workflows can be complex analysis pipelines representing in-silico experiments.

The bioinformatics community has taken up the Web Services technology: it was recommended and promoted by the EMBRACE project [2]; the ELIXIR initiative building on the EMBRACE guidelines, encourages use of Web Services in the design of distributed data infrastructure [3]; many publishers of bioinformatics tools provide SOAP interface to their software (e.g. EBI[2], CBS in Denmark[3], IBCP in Lyon[4], CBU in Bergen[5]). Several workflow management systems that make extensive use of Web services have been developed, e.g. Taverna [4], Kepler [5], Triana [6], and Sedna [7]. The SOAP Web Services have the potential to become the common digital platform for data access and exchange in bioinformatics.

However, in the era of high-throughput experiments, the technology designated to integrate distributed scientific resources must have capacity to manage massive volumes of data. XML messages used in the SOAP protocol are expensive to process, and voluminous to send over the Internet. The great advantage of having data structured in XML, parsed and loaded into memory on arrival, becomes the problem of SOAP as the data size grows. The capability of handling large-scale data with SOAP Web Services has been a concern in context of scientifc applications [8–10]. Improving the efficiency of SOAP Web Services and workflows is important for ensuring their wider endorsement in bioinformatics.

One of the key advantages of SOAP Web Services is the possibility of manipulating structured XML data, which is automatically validated and ready-to-use right after a SOAP message arrives. Hence, instead of seeking more efficient data representations and transport protocols, we aim at optimizing the resource utilization by Web Services that exchange structured data using SOAP. We approached the problem by breaking it into solvable sub-problems using a well known paradigm for algorithm design - divide and conquer. This paradigm has succesfully been applied to optimize data transport (e.g in the TCP/IP stack) and to allow processing of huge XML documents (e.g. the Pegasus [11] workflow management system breaks down XML-based specification of large worklows into pieces [12]). As we show in the results, the partitioning can largely improve SOAP performance as well. By dividing the data, we were able to run a genome-scale pipeline using standard SOAP Web Services. Our results prove that sensible use of the technology that is reported to be inefficient [8, 13], and questioned to scale [9], can bypass the limitations and enable high-throughput workflows composed of standard SOAP Web Services.

[1]`http://www.w3.org`

[2]`http://www.ebi.ac.uk/Tools/webservices/`

[3]`http://cbs.dtu.dk/services/ws.php`

[4]`http://gbio-pbil.ibcp.fr/Tools/`

[5]`http://api.bioinfo.no/wsfront/`

2 Related work

RESTful services [14] are alternative to SOAP Web Services method of integrating software systems. In contrast to message-oriented SOAP Web services, the central element in REST architecture is a resource. Resources are identified by a URIs, stored on servers, and accessed (possibly via proxies) by clients. In comparison with SOAP services, RESTful services are light-weight and good for *ad hoc* solutions, while SOAP Web Services are more advanced, featuring support for security and quality-of-service, which makes software systems more robust and reliable [15]. In this respect REST is better suited for explorative development, and SOAP for building large, long-lasting software infrastructure. Applicability of both technologies to bioinformatics is further discussed in [16].

A standard optimization method for SOAP Web Services is Message Transmission Optimization Mechanism (MTOM)[6]. MTOM attachments allow to send arbitrarily large files in a binary form, which optimizes the bandwidth utilization. The MTOM attachment is not treated as the rest of the SOAP message with respect to (de)serialization and format validation. This accounts for a great spare of resources, but comes on a cost of the main advantage of using SOAP Web Services: the data sent by an MTOM attachment is not validated, and no structure nor format can be forced on the attached file(s). The MTOM attachments are excellent for sending data in a well-established file format, which is accepted and consumed by all services. However, attachments become problematic in scenarios where data-conversion between syntactically incompatible formats is necessary, or when a subset of the data needs to be extracted on-the-fly between two service invocations.

Styx Grid Services (SGS) [17] attempt to cater for efficient data-exchange interoperable services. SGS use a file-sharing protocol, Styx, to exchange messages and facilitate streaming of data between services: a result is passed as a reference to the subsequent service, and retrieved using Styx protocol. SGS support is built-in into Taverna, and after wrapping with a SOAP Web Service wrapper, they can work with other systems as well (e.g. Triana). An idea of passing references instead of data is also used in 'handle-aware' services and Kepler [18]. The Kepler workflow engine avoids mediating the data between subsequent services in a workflow, by passing a data-handle instead, i.e. the data-producing service returns a data-handle to the Kepler workflow engine; the data-handle is forwarded to the data-consuming service, and used to retrieve the data directly from the data-producing service with a file transport protocol (e.g. FTP, SCP, HTTP). *Pass-by-reference* (or handle) approach is a 'lazy' data-transport method that can greatly spare resources, by transporting the acctual data only when necessary. Use of more efficient file-transport protocols like Styx or FTP, also contributes to the increased performance. The drawbacks of the *pass-by-reference* approach are similar to MTOM attachments (and any file-based approach): lack of automatic validation of data format, and no support for on-the-fly manipulation of the data. Also, tracking of data provenance becomes more difficult, if the data is stored by the workflow orchestrator only as a temporary reference.

Data-Grey-Box Web Services proposed in [10] is an extension to the SOAP Web Services standard that includes a middle-layer between services and service consumers. The middle-layer is responsible for storing and moving the data using specialized tools, opposed to the original standard where the data-exchange is direct and using SOAP messages. The solution, although elegantly separating data and functional concerns, has several impractical requirements: (1) an

[6]`http://www.w3.org/TR/soap12-mtom/`

extra set of mediator services (introducing additional points of failure); (2) an input- and output-data repository exposed on the client side; and (3) an extension of the Web Service standards (e.g. WSDL) that breaks existing clients.

Very recently a new W3C recommendation on a binary XML format has been published. Efficient XML Interchange Format (EXI)[7] compacts XML documents using the XML Schema[8] to guide and optimize the process, resulting in better compression than gzip[9]. EXI documents are also faster in parsing and support streaming, which will definitely have impact on performance of SOAP Web Services in the future. Currently only one commercial library provides support for efficient XML in Web Services.

3 Results

First, we present the data-partitioning communication pattern for standard SOAP Web Services, which is the main result of this work. A Java framework for implementing PartIO services is an additional result, and will be described briefly. Further, we present the outcomes of the PartIO evaluation: a *benchmark* of the communication patterns, a *genome-scale test*, and a *BPEL test*.

3.1 Data-partitioning communication pattern

Typically an execution of a procedure on an array of input elements, involves calling the procedure with the array as an argument. If the procedure is executed by a Web service, a client sends a request with an array of inputs, and receives (or retrieves) a reply with an array of outputs. All the input elements of an array are sent in one message, and all the output elements are sent in one message, thus we refer to this approach as All-In-All-Out (AIAO). An advantage of sending the entire input as one piece is that the service can find the optimal way to process it, e.g. prepare resources, process in batches, employ concurrent computations. The overhead involved in starting the computation is little, and the bandwidth consumption is also reduced as minimal overhead in network communication is needed to transfer the data, i.e. two messages are used: 1 to send input and 1 to get output. However, the throughput of the AIAO communication pattern is limited by the size of the array that can be sent in a single message. Also, partial results are not accessible before the execution is complete.

Another common approach relies on iterating over the input array and invoking the Web service operation for single input elements. Each request and each reply carry only one input and one output element, respectively. In the rest of the text we refer to this approach as One-In-One-Out (OIOO). The OIOO communication pattern has a low throughput (i.e. one input at a time), unless simultaneous calls to a Web service are enabled (called data parallelism by [19]). In contrast to the AIAO, the OIOO is not limited by the size of the input array, and it provides insight into partial results of the execution. In addition, the OIOO increases workflow performance by *overlapping communication and computation* (Fig. 1). However, the OIOO has several drawbacks:

[7]`http://www.w3.org/TR/2011/REC-exi-20110310/`
[8]`http://www.w3.org/XML/Schema`
[9]`http://www.gzip.org/`

- Numerous messages with single data elements increase the proportion of bandwidth utilized to convey message headers, rather than data.
- The overhead involved in starting an execution on the computational resource multiplies by the length of the input array, and becomes substantial if single executions are rapid and the input array long
- Multiple simultaneous invocations create separate computational jobs for each input element. These jobs often require tracking for status, in addition to sending the input and retrieving output. Such frequent communication creates a high load on a service, reduces its performance, and may cause flooding in extreme cases. The number of simultaneous jobs needs to be constrained, which significantly limits the throughput of a pipeline using OIOO communication pattern.

In the Partitioned Input/Output (PartIO) communication pattern a Web service is treated as a processor converting a stream of input data into a stream of output data. The input data is appended in partitions in separate request messages, that are correlated with one execution. The output data is retrieved in partitions by separate reply messages, as soon as it is produced. The execution is not complete before the stream of input is closed and all the input data is processed.

The AIAO and OIOO can be viewed as two extremes on the scales of message size and frequency of service invocations. The PartIO communication pattern is an intermediate approach, aiming at optimizing the balance between the message size and communication frequency. The PartIO combines the advantages of both AIAO and OIOO, and at the same time removes their limitations. Data-partitioning allows a SOAP Web Service to exchange XML data of genome-scale, split between messages of suitable size. The frequency of communication is kept low by aggregating single input and output elements into portions, and also by correlating all partitions with one execution. This way the number of requests tracking status of a job is dramatically reduced (compared to parallel OIOO). Variable partition size allows for balancing and optimizing the cost of splitting between too many messages (i.e. network and execution overhead), and the cost of processing large messages. Similarly to the OIOO, the PartIO communication pattern naturally facilitates *overlapping communication and computation*, and allows for inspection of intermediate results.

We designed an abstract Web service interface that supports communication in the PartIO pattern. We found following operations necessary:

```
// prepares the computation by sending necessary parameters and acquiring a job identifier
initJob ( job_description ) : job_id

// starts computation on the input data
startJob ( job_id ) : void

// submits a portion of data; can be invoked multiple times
appendInput ( job_id , input_element [] ) : void

// informs the service that all the input data has been sent
closeInput ( job_id ) : void

// tracks the current status of a job by informing about the number of processed input elements
// and available output elements
getStatus ( job_id ) : status

// retrieves a given number of single output elements; can be invoked multiple times
getResult ( job_id , number_of_results ) : result_element []
```

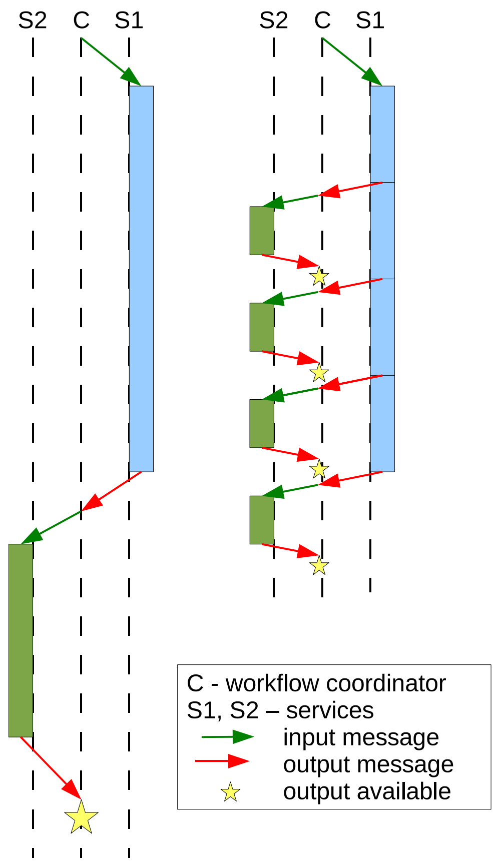

Figure 1: The effect of overlapping communication and computation. On the left-hand side the workflow coordinator (orchestrator) invokes service S1 with the entire input. After S1 completes and its entire output is retrieved, service S2 is invoked. On the right-hand side, the S1 service is invoked with the entire input, but the output is retrieved in portions as soon as they become available, and is immediately transported to S2. Transport of partial results from S1 to S2 overlaps with ongoing computations on S1 (and possibly S2). As depicted in the figure, the overlap can substantially reduce the overall run-time of a pipeline. Also partial results may become available before the pipeline completes. The OIOO and the PartIO strategies, in contrast to AIAO, make the overlapping communication and computation possible.

3.2 The PartIO framework

We developed a Java framework that facilitates development of PartIO Web Services. The framework incorporates the common logic for data-partitioning services, and in addition allows to execute computations locally, or transparently on a local cluster or the Grid. Architecture of a PartIO service built on top of the framework is presented and described in Fig. 2.

Execution of a PartIO service starts by initiating a computation (job) and creating a unique identifier of the job. The identifier is used to append partitions of input, that are correlated with the job. The PartIO framework splits the input partitions further into single elements, and persists them in a database. Next, the PartIO framework aggregates single input elements into batches, and schedules for computation using GridSAM. Sizes of batches that are scheduled for computation are independent of sizes of input partitions appended to the job, so the granularities of transportation and computation can be configured separately. When a computation on a batch of inputs is finished, the framework splits the result of computations into singletons, and persists them in a database. As soon as first result elements are stored in the database, the PartIO service client can retrieve them in partitions of arbitrary size. Anytime after creating the job, its status can be tracked using the job identifier. Status report includes progress of computation (e.g. initiated, running, completed, failed), number of input elements appended, number of input elements processed, and number of output elements retrieved.

Four aspects of the above processing are specific to the implemented service, and are not a part of the general framework (i.e. input and output data-types, and back-end computation). These aspects are specified by abstract interfaces (see Fig. 2) and implemented by the service provider:

- XMLInputSplitter - splits a partition of input into single input elements
- InternalJobCreator - creates a job specification for GridSAM and input file(s) necessary for the computation
- OutputItemCreator - creates a list of single output elements from the results of the computation
- XMLOutputAssembler - assembles single output elements into a partition of a result, which is retrieved by the client

3.3 Performance assessment

To evaluate the partitioning strategy, we have performed a series of tests on a simple gene annotation pipeline (Fig. 3). One BLAST service instance allowed search (alignment) in any of three databases: UniProtKB/Swiss-Prot [20], UniProtKB/TrEMBL [20], and NCBI NR [21]. In the following text, executions of the service using a particular database are distinguished by a subscript with the name of the database, e.g. $\text{BLAST}_{Swiss-Prot}$, BLAST_{TrEMBL}, BLAST_{NR}.

Three versions of the annotation workflow were implemented, each using one of the considered communication patterns, i.e.:

1. All-In-All-Out (AIAO) - sends all the input data in one request and retrieves all the output data in one response
2. One-In-One-Out (OIOO) - sends input and retrieves output data as single input/output elements in separate request/response messages. Each input element initiates a separate computational job.
3. Partitioned Input/Output (PartIO) - sends and retrieves input/output data in partitions (or batches of single elements)

3.3.1 Benchmark

Although we address the throughput limitation of SOAP Web Services, we carried out a small-scale test where we analyzed the performance differences arising only from the differences in communication patterns. The workflows were run on a dataset of 500 EST-contig sequences,

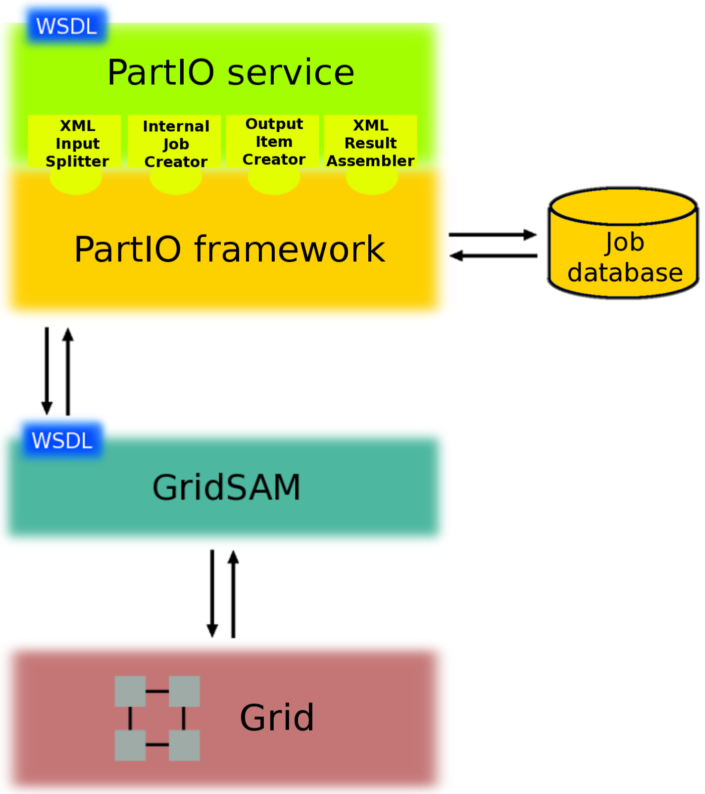

Figure 2: Architecture of a PartIO service built on top of the PartIO framework. The service implementation exposes its own WSDL interface and uses the framework for managing data-partitions. Implementation of four aspects specific to the service function (i.e. XMLInputSplitter, InternalJobCreator, OutputItemCreator, and XMLOutputAssembler) needs to be provided. The PartIO framework uses a database to persist information about jobs, and input and output data. Communication with the Grid infrastructure and monitoring computations is delegated to GridSAM. Core of the framework interacts with GridSAM via a SOAP Web Service interface, so both components can be deployed on different physical sites.

three times each. Average values for memory use (MEM), processor use (CPU), data-transport time and run-time are reported.

Overall run-times of the AIAO, PartIO and OIOO workflows, as well as the resource consumption of the Web service stack in the three approaches are summarized in Table 1. The data-partitioning workflow completed the execution in the shortest time. With respect to resource consumption, the data-partitioning communication pattern is in-between the AIAO and the OIOO, i.e. PartIO workflow required on average less memory than AIAO, but more than OIOO. On the other hand, the PartIO workflow required on average more CPU than AIAO, but less than OIOO. The time of data-transport correlated with number of messages used to transport the data, i.e. it was the shortest for AIAO and the longest for OIOO. Availability of partial results is summarized in Table 2. At any given timepoint in the workflow run-time (except at the very beginning) the PartIO workflow had in summary more input data processed than AIAO and OIOO.

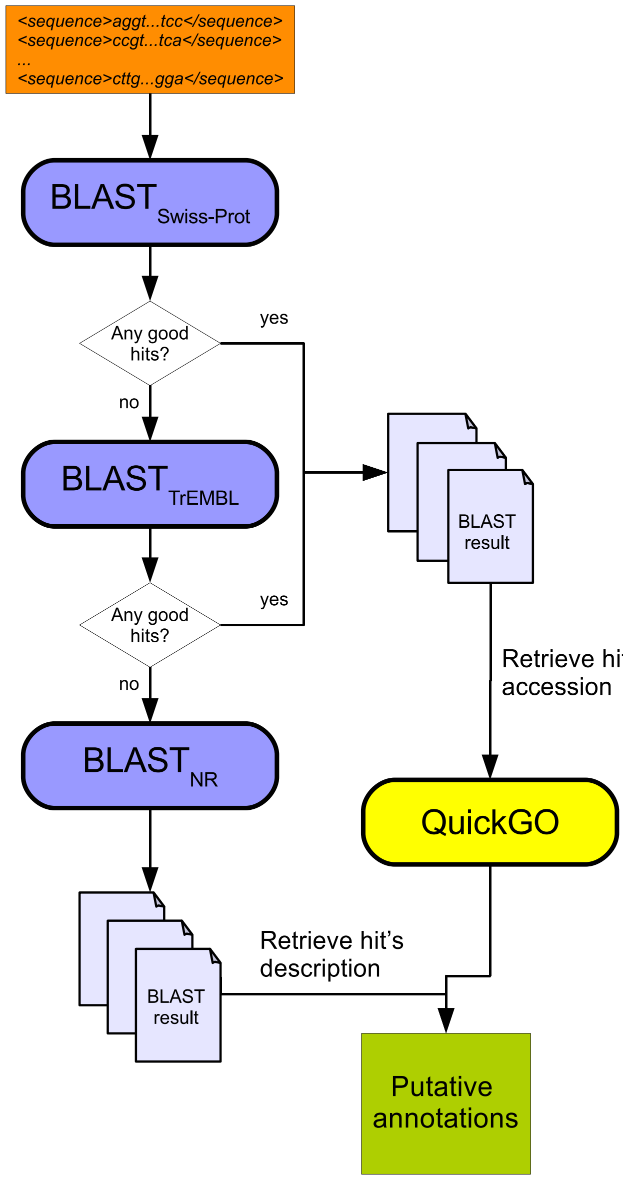

Figure 3: Gene annotation pipeline. The pipeline comprises subsequent BLAST searches in three protein databases: UniProtKB/Swiss-Prot, UniProtKB/TrEMBL and NCBI NR. First, input gene sequences are searched in Swiss-Prot. If no hits is found, TrEMBL is searched. In case of no match, NR is used. Next, Gene Ontology annotations for hits from Swiss-Prot and TrEMBL are retrieved using QuickGO, and used as annotations for the input genes. Genes that only have similar sequences in NR, get the description of the most similar sequence as a putative annotation.

3.3.2 Genome-scale test

To test the throughput-limits of the considered communication patterns and their applicability to large-scale analyzes, we executed the annotation workflows on a genome-scale dataset. In every workflow run, in total 23k alignment searches were performed: EST-contig sequences from the 14k input dataset that didn't get significant hits in BLAST$_{Swiss-Prot}$, were searched in BLAST$_{TrEMBL}$, and then if failed to align again, in BLAST$_{NR}$ (see Fig.3). The computation ran on a large cluster and exploited available computational resources to the highest degree feasible. The AIAO workflow was excluded from this comparison due to its limitation on the

Table 1: Resource utilization in the benchmark. The AIAO, OIOO, and PartIO communication patterns were compared with respect to: processor (CPU) and memory (MEM) utilization during message processing by the Web service stack; data-transport time; and the overall run-time of the workflow. The last column presents percentage difference in run-time relative to the PartIO workflow run-time.

	CPU [%]	MEM [%]	Data-transport time [s]	Run-time [s]	Run-time relative to PartIO [%]
AIAO	11.87	33.66	16.4	6591	127
OIOO	46.59	19.34	43.7	5826	112.5
PartIO	19.98	25.91	19.5	5180	100

Table 2: Availability of partial results in the benchmark, presented in percent of the total result available. The availability of partial results is compared for the three workflows, in the three main steps of the pipeline: BLAST$_{Swiss-Prot}$ (SP), BLAST$_{TrEMBL}$ (TR), BLAST$_{NR}$ (NR). The availability is shown at three checkpoints of the workflow run-time. The AIAO workflow does not provide insight into the final results until the end of the pipeline. At each stage (e.g. BLAST$_{Swiss-Prot}$, BLAST$_{TrEMBL}$, BLAST$_{NR}$), only the entire result from the stage is made available at the end. In contrast, the OIOO workflow was able to provide as much as 10% of the final results in 1/4 of the pipeline run-time, and 32% final results when it was half-way. The PartIO workflow produced first final results in the middle of the run-time, but at that time it had the entire BLAST$_{Swiss-Prot}$ stage completed, and 79% of the BLAST$_{TrEMBL}$ stage.

	1/4			1/2			3/4		
	SP	TR	NR	SP	TR	NR	SP	TR	NR
AIAO	100	0	0	100	0	0	100	100	0
OIOO	31	18	10	51	41	32	74	67	58
PartIO	100	12	0	100	79	1	100	96	44

data size, which was significantly lower than size of the input dataset.

The PartIO workflow completed the annotation pipeline on 14k sequences in 3 hours and 20 minutes. The OIOO workflow required 23 hours and 28 minutes to annotate the same dataset. The average throughput measured in number of query sequences scheduled for processing in parallel in the PartIO workflow was 3365, compared to 85 in the OIOO workflow (Fig. 4). The maximum throughput reached by the PartIO workflow was 13767 query sequences scheduled for processing at the same time. Both workflow executions had spare computational power available on the cluster (over 200 free CPUs were available during both runs).

Processor and memory utilization was measured for the Web service stacks of the BLAST service and of the workflow orchestrator. The measurements are presented in Figure 5 and in Table 3. The Web service invoked in the PartIO pattern used less than half of the CPU power required by the OIOO Web service (18.5% compared to nearly 42%). The PartIO service had also 25% lower memory consumption, i.e. 16.46% for PartIO and 22.1% for OIOO. Only small difference was observed in resources used by the workflow orchestrators. Sum of the time spent on transporting the input and the output data was 280.3 seconds for the PartIO workflow and 1005.6 seconds for the OIOO workflow.

For every BLAST step of the PartIO workflow run, we calculated the back-end execution time of the BLAST tool. Sum of the times required to complete each of the three BLAST steps was

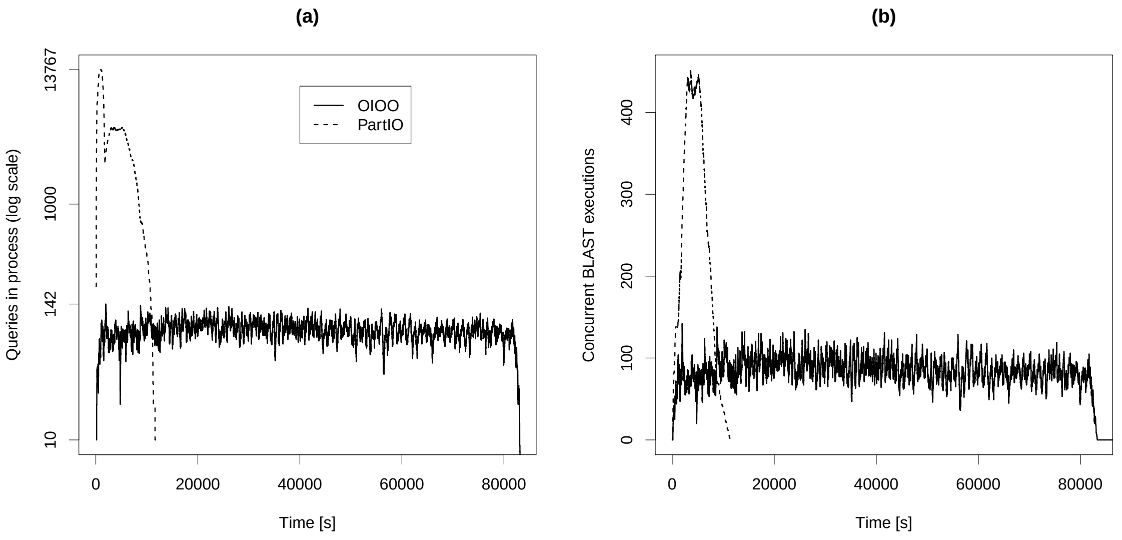

Figure 4: Throughput of the BLAST service invoked in the PartIO and OIOO communication patterns in the genome-scale test. (a) Workflow throughput measured in number of query sequences being scheduled concurrently for similarity search (logarithmic scale on Y axis). In the OIOO workflow the average throughput was 85, with a maximum at 142. In the PartIO workflow the average throughput was 3365, with a maximum at 13767 (b) Throughput measured in concurrent executions of the BLAST program on the cluster. Each BLAST execution corresponds to one CPU used on the cluster. In the PartIO workflow, the BLAST service used on average 218 CPUs (max 452 CPUs). In the OIOO workflow, the BLAST service used 85 CPUs on average (max at 142). The numbers are equal to the query-sequence measure of throughput (a), since OIOO services process one input sequence per execution.

Table 3: Resource utilization in the genome-scale test. The OIOO and PartIO communication patterns were compared with respect to average processor (CPU) and memory (MEM) utilization during message processing by the Web service stack (ws), and by the workflow orchestrator (wf). Sum of the time spent on communication and the overall run-time of the workflows is also presented.

	CPU [%]		MEM [%]		Data-transport	Run-time
	(wf)	(ws)	(wf)	(ws)	time [s]	[hrs]
PartIO	1.84	18.47	8.52	16.46	280.3	03:20
OIOO	1.64	41.99	9.99	22.10	1005.6	23:28

44 seconds longer than the runtime of the entire PartIO pipeline. It means that if the analysis was executed directly on the cluster, running the three BLAST searches one after another, it would take longer than the same analysis using SOAP Web Services.

3.4 BPEL test

This test was a proof-of-principle experiment aiming to show that by using PartIO, the standard Web technologies can be used to perform a genome-wide analysis. Web standards and interoperability are central to our work. We designed the PartIO communication pattern following the WS-* standards and OASIS Web Service Interoperability (WS-I) guidelines, to make sure it will be possible to use it in any service-oriented setting. The *BPEL test* also proves that the

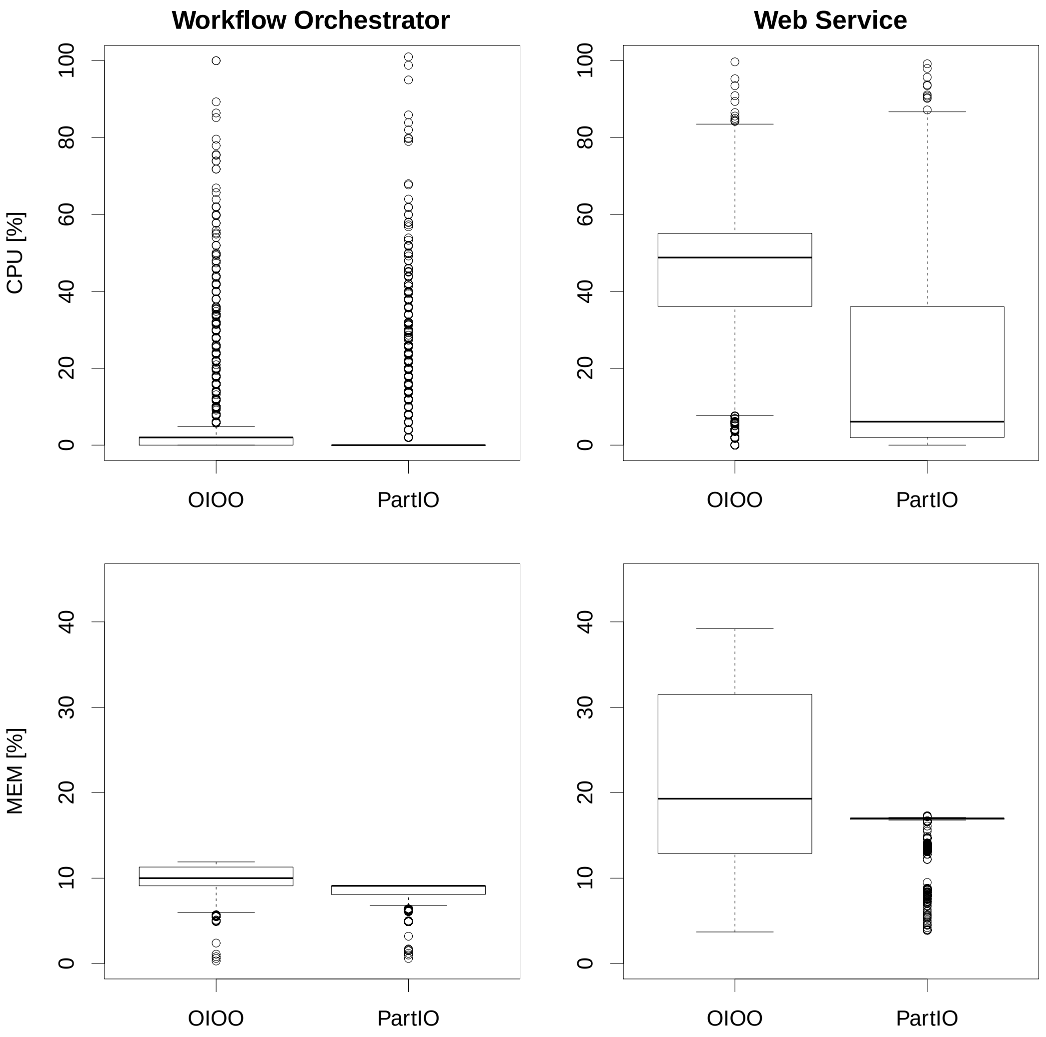

Figure 5: Resource utilization in the genome-scale test. The OIOO and PartIO communication patterns were compared with respect to processor (CPU) and memory (MEM) utilization during message processing by the Web service stack, and by the workflow orchestrator. The boxplots present distribution of measured values during workflow executions: the smallest and the largest value (lower and upper whisker, respectively), lower and upper quartile (bottom and top of the box, respectively), and median (bar in the box). Outliers are presented as circles. Measurements were taken every second.

PartIO services conform to standards and can be orchestrated by standard SOAP Web Service orchestration tools.

We implemented the gene annotation pipeline (Fig. 3) in BPEL[10] - a *de-facto* standard Web Service orchestration language. The workflow was deployed on a regular desktop in an out-of-the-box workflow engine, and was used to annotate 17k contig sequences from a transcriptome assembly of honey bee (*Apis mellifera*). The entire execution required 14 hours and 15 minutes to finish.

[10]`http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`

4 Discussion

Our tests compared the data-partitioning communication pattern with typical patterns of interaction with SOAP Web Services in a bioinformatic workflow. The AIAO strategy is common in bioinformatic services, which are often used to batch-process arrays of objects (e.g. services predicting post-translational modifications of proteins deployed at CBS, Technical University of Denmark). AIAO services are simple in design and use, and in most cases are perfectly sufficient. However, as proven by the *genome-scale test*, they do not scale to facilitate use in high-throughput analysis pipelines.

The OIOO communication pattern is probably the most prevalent among bioinformatics services (e.g. sequence similarity search services deployed at EBI in Hinxton). OIOO does not intrinsically support large-scale computations, but by using concurrent invocations, the throughput of an OIOO service can be largely improved. The Taverna workflow management system exploits this potential in *intra-processor data parallelism* for implicit iterations [22]. Nevertheless, the extent of throughput gain achieved by parallelization of service invocations is limited by the number of concurrent requests a service is able to handle. Too large frequency of invocations may result in a service spending more time serving the communication than processing the data, and in extreme cases may even overload the service (e.g. Taverna controls the number of concurrent invocations to a service with a parameter). For Web services with access to a computational resources with a large capacity, the OIOO interface is a throughput bottle-neck. This is clearly illustrated by the comparison of OIOO and PartIO workflows in the *genome-scale test* where very high CPU usage of the BLAST service invoked in OIOO pattern indicates heavy load caused by frequent communication. The heavy load hindered full utilization of computational resource which resulted in significantly lower throughput and over 7 times longer run-time.

The data-partitioning communication pattern lifts the limitation on the throughput of SOAP Web Services. Flexible balancing of the number of requests and sizes of data partitions enables a Web service to spare resources (memory, CPU, data-transfer time) and increase throughput. The stream-like interface of PartIO services, allows to overlap communication and computation (pipelining effect), and hence reduces the run-time. It is observed in the *genome-scale test* where the execution time of the analysis using PartIO Web Services was shorter then the execution time of the same analysis performed step-by-step directly on the cluster.

Other approaches that aim at improving efficiency of Web services focus on the transport layer, either optimizing the data-transfer (MTOM, Data-Grey-Box Web services), or reducing it to the necessary minimum ('pass-by-reference', i.e. SGS, data-handles). The data-partitioning communication pattern does not reduce the volume of data that is exchanged, but allows to balance the load, making handling of huge data feasible. The performance improvement is a direct result of the pipelining effect, not present in any of the proposed approaches. Additional contrast with the Data-Grey-Box Web services is that the data-partitioning pattern does not require any change to the SOAP Web Service standards, and can make use of any of existing Web service tools.

PartIO-enabled optimizations apply to workflow orchestrators as well, and allow use of standard Web Service orchestration tools. As a proof of concept we annotated a deep-sequencing (RNA-seq) dataset executing our pipeline in an out-of-the-box BPEL workflow engine. BPEL was designed for business applications, and has no special support for scientific needs, in particular large data-traffic [23]. The execution time elapsed by the BPEL workflow was longer than the PartIO Java workflow used in the *genome-scale test* test, but the BPEL engine performs many additional operations (e.g. stores all messages and communication history in a database) which were not part of the Java workflows in the *genome-scale test*. Taking that and the larger input dataset into consideration, the difference in run-time when compared to the OIOO workflow is substantial. The BPEL workflow example indicates that the standard Web technologies are powerful enough to fully facilitate *in-silico* experimentation on a large scale.

The data-partitioning communication pattern is independent of the back-end computational resources, but aims at maximizing throughput of complex computational pipelines, and therefore it is natural to discuss (and test) it in the context of high-performance computing (HPC) infrastructures. In HPC, task fragmentation (or aggregation) is a common optimization strategy. PartIO services with access to HPC resources, can make use of the data-partitioning performed on a Web service communication level, bring it down to the computing level, and further optimize load on the Grid (similarly to data chunking in [24]). In this respect, the data-partitioning has an added value in Grid settings.

In the *benchmark* we focused on the influence of the partitioned communication on the Web service stack, and its impact on the entire workflow run-time. With respect to resource consumption, the data-partitioning communication pattern was ranked between the two opposite extremes of communication frequency and message size, i.e. AIAO and OIOO. The total workflow run-time was however shortest for the PartIO workflow, which shows that correct balance of the communication granularity together with the pipelining effect optimized the workflow performance. Moreover, we believe that the merits of the partitioned communication were reduced by a fast network and the choice of the test workflow, i.e. the overall communication time in the pipeline constituted less than 0.4% of the average workflow run-time in case of the PartIO workflow, and 0.75% for the OIOO. We expect even larger differences in performance in circumstances where data transportation accounts for a more significant share of the workflow run-time.

5 Conclusion

We have presented a partitioned input and output pattern for communication between workflow orchestrator and SOAP Web Services. We have shown that the data-partitioning lowers the resource demand of standard SOAP Web Services. In consequence, performance and throughput of the services and workflows is largely improved, and makes the standard SOAP Web Services capable of executing genome-scale *in silico* experiments without a significant overhead.

The standard Web technologies were proposed to connect the distributed resources in bioinformatics due to their ability to communicate across boundaries (of platforms, languages and networks). In addition, they have a potential to make *in silico* analysis faster, more robust, accessible and reproducible. We have shown that, if used sensibly, the standard SOAP Web Services can address the needs of genome-scale research. The PartIO communication pattern together with the software framework that we have developed, will hopefully ease implementation of high-capacity services, and in effect trigger wider adoption of the SOAP Web Services in bioinformatics.

In the light of new developments of the W3C standard for binary XML exchange, EXI, we plan to include streaming of binary XML documents into the PartIO framework. Integration with other approaches to improve performance of standard SOAP Web Services and workflow systems will also be considered.

Acknowledgements

We would like to thank Inge Jonassen for his feedback regarding this work and the manuscript. We also acknowledge the entire eSysbio team at the Computational Biology Unit and Parallab in Bergen for useful comments and interesting discussions.

Funding: eSysbio project (http://esysbio.org) funded by The Research Council of Norway (eVita program, project nr. 178885).

References

[1] L. Stein. Creating a bioinformatics nation. *Nature*, 417(6885):119–120, 2002.

[2] S. Pettifer, J. Ison, M. Kalaš, D. Thorne, P. McDermott, I. Jonassen, A. Liaquat, J.M. Fernández, J.M. Rodriguez, INB-Partners, D.G. Pisano, C. Blanchet, M. Uludag, P.M. Rice, E. Bartaseviciute, K. Rapacki, M.L. Hekkelman, O. Sand, H. Stockinger, A.B. Clegg, E. Bongcam-Rudloff, J. Salzemann, V. Breton, T.K. Attwood, G. Cameron, and G. Vriend. The embrace web service collection. *Nucleic Acids Research*, 38:683–688, 2010.

[3] A. Bairoch, M. Ashburner, L. Bougueleret, V. Breton, and S.A. Sansone. Elixir wp 7 – data integration and interoperability. Technical report, ELIXIR WP7, 2009.

[4] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M.R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(Web Server issue):729–732, 2006.

[5] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. In *16th International Conference on Scientific and Statistical Database Management, 2004. Proceedings.*, pages 423–424, 2004.

[6] I. Taylor, M. Shields, I. Wang, and A. Harrison. The triana workflow environment: Architecture and applications. *Workflows for e-Science*, pages 320–339, 2007.

[7] B. Wassermann, W. Emmerich, B. Butchart, N. Cameron, L. Chen, and J. Patel. Sedna: A bpel-based environment for visual scientific workflow modeling. *Workflows for e-Science*, pages 428–449, 2007.

[8] R.A. van Engelen. Pushing the soap envelope with web services for scientific computing. In *International Conference on Web Services*, pages 346–352, 2003.

[9] P.B.T. Neerincx and J.A.M. Leunissen. Evolution of web services in bioinformatics. *Briefings in Bioinformatics*, 6(2):178–188, 2005.

[10] D. Habich, S. Preissler, W. Lehner, S. Richly, U. Assmann, M. Grasselt, and A. Maier. Data-grey-box web services in data-centric environments. In *International Conference on Web Services*, pages 976–983, 2007.

[11] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In *European Across Grids Conference*, pages 11–20, 2004.

[12] E. Deelman. Grids and clouds: Making workflow applications work in heterogeneous distributed environments. *International Journal of High Performance Computing Applications*, 24:284–298, 2010.

[13] K. Chiu, M. Govindaraju, and R. Bramley. Investigating the limits of soap performance for scientific computing. In *Proceedings 11th IEEE International Symposium on High Performance Distributed Computing*, HPDC '02, pages 246–254. IEEE Computer Society, 2002.

[14] R.T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[15] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. 'big' web services: making the right architectural decision. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM.

[16] H. Stockinger, T. Attwood, S.N. Chohan, R. Côté, P. Cudré-Mauroux, L. Falquet, P. Fernandes, R.D. Finn, T. Hupponen, E. Korpelainen, A. Labarga, A. Laugraud, E. Pafilis, M. Pagni, S. Pettifer, I. Phan, and N. Rahman. Experience using web services for biological sequence analysis. *Briefings in bioinformatics*, 9(6):493, 2008.

[17] J.D. Blower, A.B. Harrison, and K. Haines. Styx grid services: Lightweight, easy-to-use middleware for scientific workflows. *Computational Science - ICCS 2006*, pages 996–1003, 2006.

[18] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[19] T. Glatard, J. Montagnat, and X. Pennec. Efficient services composition for grid-enabled data-intensive applications. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC'06), Paris, France*, pages 333–334. IEEE Computer Society, 2006.

[20] M. Magrane and UniProt Consortium. UniProt Knowledgebase: a hub of integrated protein data. *Database*, 2011:bar009, 2011.

[21] K.D. Pruitt, T. Tatusova, W. Klimke, and D.R. Maglott. Ncbi reference sequences: current status, policy and new initiatives. *Nucleic Acids Research*, 37(Database issue):32–36, 2009.

[22] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble. Taverna, reloaded. In *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management*, SSDBM'10, pages 471–481. Springer, 2010.

[23] D. Habich, S. Richly, S. Preissler, M. Grasselt, W. Lehner, and A. Maier. Bpel-dt - data-aware extension of bpel to support data-intensive service applications. In *European Conference on Web Services*, 2007.

[24] V.S. Kumar, P. Sadayappan, G. Mehta, K. Vahi, E. Deelman, V. Ratnakar, J. Kim, Y. Gil, M.W. Hall, T.M. Kurç, and J.H. Saltz. An integrated framework for performance-based optimization of scientific workflows. In *IEEE International Symposium on High Performance Distributed Computing*, pages 177–186, 2009.

[25] A.A. Adzhubei, A.V. Vlasova, H. Hagen-Larsen, T.A. Ruden, J.K. Laerdahl, and B. Høyheim. Annotated expressed sequence tags (ests) from pre-smolt atlantic salmon (salmo salar) in a searchable data resource. *BMC Genomics*, 8:209, 2007.

[26] K.R. von Schalburg, M.L. Rise, G.A. Cooper, G.D. Brown, A.R. Gibbs, C.C. Nelson, W.S. Davidson, and B.F. Koop. Fish and chips: various methodologies demonstrate utility of a 16,006-gene salmonid microarray. *BMC Genomics*, 6:126, 2005.

A Methods

A.1 Implementation

The PartIO framework was implemented using Java 1.6. The core of PartIO uses a relational database for storing information about jobs, input and output data (see Fig. 2). The database records are accessed and manipulated using Apache OpenJPA 1.2.1 persistence library. The PostgreSQL 8.3 database was used in the tests. The framework delegates job submission to GridSAM[11] 2.3.0, which is an application providing a job submission interface to many commonly used distributed resource management systems. The communication uses GridSAM's Web service interface, and JSDL[12] for job description. GridSAM was configured to use TORQUE (formerly PBS) for job scheduling on a cluster.

In Web service and workflow development for the *benchmark* we used Java 1.6, Apache Axis2 1.4 as a SOAP stack, and XMLBeans 2.3 for binding XML to Java code. In the *genome-scale test*, Apache Axis2 1.5.4 and XMLBeans 2.4 were used. In the *BPEL test* the workflow was implemented in BPEL language and deployed in Apache ODE 1.3.5 workflow engine. All tests services and workflows (except BPEL workflow) were developed in Java 1.6, and services were deployed in the Apache Tomcat 6.0.32 container.

[11]`http://www.omii.ac.uk/wiki/GridSAM`
[12]`http://forge.gridforum.org/projects/jsdl-wg`

A.2 Hardware setup

The test workflows were run on a computer with 4GB of memory and a Dual-Core 2.8GHz processor. The BLAST Web service using PartIO framework (Fig. 2) was deployed on a virtual machine with 3GB of memory and a Dual-Core 2.6GHz processor. The actual computation by the BLAST tool was run on a cluster: in the *benchmark* a dedicated queue consisting of 12 cores was used (3 identical nodes with 2 dual core processors and 4GB of memory each), in the *genome-scale* and *BPEL tests* a common queue with 164 heterogeneous nodes and 846 processors was used. All computers shared a 100Mbit network.

A.3 Gene annotation pipeline

The pipeline is inspired by the automatic EST annotation pipeline described in [25]. In search for function of an unknown gene (or protein), its sequence is aligned with databases of gene/protein sequences with known function. If orthologous gene (protein) is found, it is likely that its function is similar to the function of the unknown gene (protein). In the pipeline we implemented (Fig. 3), genes were annotated by searching in three protein databases: UniProtKB/Swiss-Prot [20], UniProtKB/TrEMBL [20], and NCBI NR [21]. NCBI BLAST program was used to perform the serach/alignment. The order of databases in the pipeline corresponds to the quality of information they contain, and their size. First, unknown genes were aligned with Swiss-Prot (Feb 2011 release) which was the smallest (280 MB) and contained 525,207 manually curated records. If no significant matches were found (e-value cutoff at 1e-15), the gene sequence was searched against TrEMBL (Feb 2011 release) that contains automatically annotated data (6.5GB, 13,499,622 records). In case of no hits, the last search was performed in NR (April 2011 release), which contained 13,841,106 sequences (9.8GB) with low quality annotations. For the hits found in SwissProt and TrEMBL, Gene Ontology terms were retrieved using QuickGO [13], and they were used to annotate input gene sequences. For those genes that matched NR records, description of the sequence record was used as a putative annotation.

A.4 Input datasets

The input dataset for the *genome-scale test* consisted of 13767 unique EST-contig sequences from the cGRASP 16k microarray [26]. For the *benchmark* we used a subset of the cGRASP dataset, containing 500 contig sequences. In the *BPEL test* we annotated RNA-seq reads mapped to 17182 contigs of the transcriptome assembly being part of the NCBI *Apis mellifera* genome build 4.1.

A.5 Test services and workflows

The test workflows implemented the gene annotation pipeline shown in Fig. 3. The BLAST service was invoked and provided with data in different patterns:

[13] `http://www.ebi.ac.uk/QuickGO/WebServices.html`

- AIAO (All-In-All-Out) - the entire input of the service is sent in one SOAP message, and the entire output data is received in one SOAP message. All the input and output is corellated with one execution (job).
- OIOO (One-In-One-Out) - input is sent in iterations, sequence by sequence. One input sequence is sent in one SOAP message and corellated with one execution (job). The output is received in iterations, BLAST result after BLAST result. One BLAST result is sent in one SOAP message. Sending input and receiveing output is concurrent.
- PartIO (Partitioned Input and Output) - input and output are sent in partitions (or batches). Every partition is sent in one SOAP message. All the partitions of the input dataset are corellated with the same service execution (job).

The test workflows are named after the communication pattern they use, i.e. AIAO workflow, OIOO workflow and PartIO workflow.

The test workflows used the same instance of a PartIO BLAST service. We measured the overhead of the PartIO framework to be smaller than 1% of overall workflow run-time (benchmark), and decided to not implement separate services for the AIAO and OIOO strategies. In this way the difference between the workflows were limited to the communication pattern, and implementation-dependent factors were excluded.

A.5.1 Benchmark

The aim of the *benchmark* was to analyze the performance of the workflows and services, coming solely from the differences in the communication patterns. Each benchmark test was repeated 3 times to assess variability of results. The AIAO and OIOO workflows invoked the services as in the description above. The PartIO workflow sent input sequences to all BLAST services in portions. The services were supplied with between 3 and 20 input sequences in one call. Similarly between 3 and 20 BLAST result documents were retrieved from the services in every message. The first BLAST search in the workflow (Swiss-Prot) was supplied with larger input messages (5 messages, 100 sequences each), since all the input was available at the start.

In order to complete the test we had to introduce an additional frequency constraint on the OIOO workflow: to avoid overflow of communication requests to a service, the number of active job requests and the frequency of polling for status had to be balanced against the communication handling capacity of the service. The limitation did not affect the throughput of the computation.

A.5.2 Genome-scale test

In the *genome-scale test* the OIOO and the PartIO workflows were compared on a genome-sized input. The aim of the test was to measure throughput, identify limitations and compare the workflows in a real-life scenario. The PartIO workflow used the following partition sizes: 20 input sequences and 20 output BLAST result documents for searches in TrEMBL and NR, and 200 input and 50 output portions for Swiss-Prot. The OIOO workflow used the same settings as during the *benchmark*, except that the limitation on concurrent active jobs was raised to 400.

The AIAO workflow was excluded from this test due to its limitation on the data size: BLAST results for 14k input sequences constituted 1.6GB of textual XML data, which when parsed would require considerably more memory than the available 3GB.

A.5.3 BPEL test

The *BPEL test* aimed to prove that the PartIO services conform to SOAP Web Services standards, and that by using PartIO strategy, standard Web technologies can be used to perform a genome-wide analysis. In this test the annotation pipeline (Fig. 3) was implemented using the BPEL workflow orchestration language. The input sequences to align with Swiss-Prot were sent in one request, but results were retrieved in portions of size 50. Subsequent searches (i.e. in TrEMBL and NR) got input and returned output in partitions of size 10 to 20. The computational hardware was identical as in the *genome-scale test*.