A Framework for Mathematical Optimization in Microservice Architectures

Stefan Guericke
stefan.guericke@maersk.com

Andrea Cassioli
andrea.cassioli@maersk.com

A.P. Moller Maersk, Esplanaden 50, Copenhagen, Denmark

In the last years, the gap between solution methods in literature and optimization running in production has increased. Agile development practices, DevOps and modern cloud-based infrastructure call for a revisit of how optimization software is developed. We review the state-of-the-art, propose a development framework that can be applied across different programming languages and modeling frameworks and test it with a relevant optimization use-case. The framework enables optimization experts to focus on solution methods while having the tools to deploy a scalable and reusable optimization application using the latest technologies and thus, increase the acceptance in industry.

Key words: mathematical optimization, microservices, cloud computing

1. Introduction

Digital technologies are changing business models and add value-producing opportunities, see Gray and Rumpe (2015). Among others, advances in data platforms, artificial intelligence and cloud systems lead to more data-driven decision making processes in various industries, see Ustundag and Cevikcan (2017). With the increased focus on data-driven decision making we expect the relevance of analytical software such as mathematical optimization to be further adopted in the future. However, it is crucial to shift optimizations beyond prototypes so that they can realise real value in industry.

With a shift to private and public cloud infrastructure, microservice architectures (MSA) became a prominent development approach, see Fowler (2014) and Di Francesco et al. (2017). In contrast to monolithic applications, the microservice approach consists of loosely coupled services providing the advantages of development agility, scalability and resilience, see Dmitry and Manfred (2014) and Amazon (2019).

These technical processes are closely interlinked with methodological processes, Di Francesco et al. (2017), and result in changes to development methods and enterprise processes. Agile and DevOps (combination of development and operations) practices empower teams to deliver end-to-end solutions and, opposed to practices in classical large cooperations, are making them responsible to run and maintain the application.

The technological and organizational shifts imposes new engineering challenges to optimization experts, often missing experience in software engineering. This results in inefficient hand-over procedures and black-boxes for part of the development teams. In practice, we have observed that a lot of emphasis has to be placed on developing software infrastructure to enable optimization, thus distracting from solution methods, iterative improvements and thorough assessments and evaluations of results.

The contribution of this paper is threefold: We review the state of the art, suggest a framework that simplifies the development process and thereby increase the applicability of optimization in industry and research and showcase its applicability to a classic optimization problem.

The remainder of this paper is structured as follows: Section 2 provides software development and engineering background to motivate the proposed architecture in this work. Section 3 reviews the state-of-the-art with respect to mathematical optimization, web-services and frameworks supporting the development process. Section 4 provides an overview of the proposed service architecture and the data and information flow. Section 5 presents a use-case to show the applicability of the framework to the bin-packing problem. Finally, Section 6 concludes the paper.

2. Background

In the introduction we highlighted the significant industry changes over the last years in how to develop, deploy and run applications. This section provides an overview of the current state-of-the-art of software engineering and enabling technologies to provide context and motivation for the derived architecture.

According to VersionOne (2019), the vast majority of enterprises uses agile or lean development methods today. Agile software development emphasises the "importance of individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation [and] responding to change over following a plan" (Beck et al. (2001)). Within agile methodologies, Scrum is the most widely used practice, see VersionOne (2019). Scrum works in iterative cycles, called sprints, in which an increment to a (software) product is developed. Within a sprint of 2 – 4 weeks, the development team of a few individuals focuses on a software increment that is demonstrable to and usable by the user. For further details on agile and Scrum, we refer to Schwaber (2004).

Closely related to agile methods are DevOps practices, which are becoming increasingly relevant, see VersionOne (2019). They aim to integrate development and operations, historically separated in large enterprises, by using technologies to automate large parts of software testing, deployment and monitoring, see Ebert et al. (2016). DevOps supports fast delivery cycles necessary for running sprints, avoids miscommunication and overhead between development and operations teams and leads to higher productivity, Virmani (2015). It empowers development teams to be responsible to develop, run and maintain the solution (including mathematical optimizations), depending on the use-case with a 24/7 service-level. Thus, it is in the team’s intrinsic interest to develop solutions that are robust, of high quality and can be changed and deployed quickly.

Due to the requirement of frequent and fast software changes microservice architectures became the predominant pattern since 2010, see Hoda et al. (2018). In contrast to monolithic applications (Figure 1a), the microservice approach (Figure 1b) consists of loosely coupled services providing advantages in development agility, scalability and resilience, see Dmitry and Manfred (2014) and Amazon (2019). Besides, they offer the ability to isolate work of different teams from each other, reuse services across different projects and scale development organizations.

To support agile, DevOps and microservices practices, several development processes and concepts have emerged and according to VersionOne (2019) are now widespread in practice: Continuous integration (CI), continuous delivery (CDE) and continuous deployment (CD). CI systems automatically validate source-code changes in version control systems before deploying the application and operate as an automated quality assurance system. It is a key technology to enable agile development and fast deployments, see Stolberg (2009). The packaging of the software, so that it can be deployed to some infrastructure either manually or automatically, is referred to as continuous delivery (CDE), Shahin et al. (2017). Continuous deployment goes one step further by automatically deploying each change to a (production) environment. With an entire CI/(CDE/)CD pipeline, every code change is automatically deployed to production without any user interaction. We refer to Duvall et al. (2007) and Shahin et al. (2017) for further details on these systems.

Several technologies such as severless computing are supporting agile, DevOps, microservices and continuous software engineering. Today, there are basically two different approaches for serverless execution environments: Function as a service (FaaS) and container orchestrators, whereas some FaaS offerings can also execute containers. Containers, such as Docker (Docker (2019)) and rkt (Red Hat (2019)), serve as an isolation technology to run individual services and provide a modern packaging for deployment. FaaS offerings (e.g. Azure Webapps, AWS Fargate or Google Compute Engine) run source code or containers and thus, decrease the maintenance effort. Often, cloud providers use container orchestration tools as their runtime for FaaS. However, FaaS is often limited in terms of memory consumption and runtime, making them less applicable for optimization use-cases.

The alternative approach of using container orchestrations (e.g. Docker Swarm, Apache Mesos and Kubernetes) also provide an environment to deploy, execute, discover and monitor multiple containers. Container orchestrators have the advantage of not being bound to artificial resource limitations. Since the setup and maintenance of orchestrators is a time-consuming task and often requires separate teams to manage, most providers started to offer managed software (as platform as a service, PaaS), where the virtual machines or physical computers are entirely managed by the providers, thus continuously decreasing the efforts involved with managing the clusters. We refer to Hightower et al. (2017) for further details on container orchestration.

Lastly, an important factor in modern development is the raise of cloud technology at scale. Nowadays cloud providers not only provision cheap storage and virtually infinite computational power, but a wide range of services from specialized storages (SQL, No-SQL and cache databases), data streaming, telemetry services, etc. Managing these resources in a consistent and scalable way requires proper tooling and process, that can be again inspired by the DevOps world (for instance using infrastructure as code approach.)

To summarize this section, we outlined how technology supports emerging agile development methods and practices and vice versa. It becomes apparent that they are closely interlinked and enabling each other. Although these practices emerged from software engineering, we observe they are becoming increasingly important for mathematical optimization and other fields, such as forecasting and machine learning.

3. State of the Art

Literature on how to deploy mathematical optimizations, in particular as webservices to cloud-environments, is scarce.

Several software frameworks exist to ease the development of some optimization classes. Those frameworks exist for a large variety of programming languages to abstract mathematical models from specific solver implementations, see for instance PuLP Mitchell (2019) for

Python, JavaILP Lukasiewycz (2008) and JuMP Dunning et al. (2017) for Julia. They simplify evaluation of different commercial and open-source solvers without reimplementing the model for a specific solver and, depending on the framework, speed up the model implementation. However, none of the frameworks provide a mechanism to deploy and run optimizations.

Smirnov et al. (2016) propose a commercial platform that solves mathematical models using the AMPL modeling and programming language on distributed systems. Besides, commercial solver vendors have started to provide cloud offerings, solving optimization models in the cloud and thus the need to provision local compute resources (see for example Gurobi (2019) and IBM (2019)). The supported optimization classes are constantly extended but often focuses on linear programming and a few non-linear classes. Custom optimization algorithms, e.g. heuristics, are only supported through AMPL, limiting the applicability of this technology. Besides, it is still unclear how the application, creating and submitting the models should be structured and interact with end-users.

Little work has been published on deploying optimizations as webservices. An early work on distributed optimization services is Czyzyk et al. (1998), the "The Network-Enabled Optimization System" (NEOS). NEOS is an internet based general purpose optimization service, supporting various optimization classes, e.g. linear, non-linear and mixed-integer programs. It evolved to now also support XML based application programming interfaces and is designed around the central server paradigm as outlined in Figure 2a. The central server distributes incoming requests to different solver servers.

Fourer et al. (2010) suggest an alternative design of how to implement optimization as a service and provide an open-source implementation as part of the COIN-OR framework that can solve a large variety of optimization classes. The architecture uses web services and derive several protocols of the service: OSiL for the optimization service problem instances (such as variables, constraints etc), OSrL to specify the optimization results and OSoL for the solver options. Using the discover language OsDL, a client communicates with the registry servers to identify and discover the appropriate solver and initiate a peer-to-peer communication with it. Thus, it follows a service-oriented architecture as shown in Figure 2b. Compared to the central server approach, their architecture is assumed to scale better, because the clients interact directly with the appropriate server. A similar approach is taken by Steglich (2016). It remains to be explored how the system creating and submitting the mathematical optimization should be structured and deployed.

Sheikhalishahi et al. (2012) state the importance of using cloud based infrastructure and platform as a service for operations research and argue that existing frameworks such as Czyzyk et al. (1998) and Fourer et al. (2010) can be provisioned on cloud infrastructure.

The review indicates that literature focusing on deploying mathematical optimization to production systems is scare and very limited work has been done on deriving an architecture pattern of how to deploy optimization solutions in the context of modern technology stacks and development practices. Based on this literature review we refine our contribution as follows:

1. Review the state-of-the-art in running mathematical optimization as services and assess their applicability to modern private and public cloud based environments using microservice architectures.

2. Provide a framework for optimization experts to simplify the process of deploying their models to customers, collaborators and other researchers, thus, enable them to focus on model building, improvement and detailed numerical evaluation.

3. Increase the applicability and spread of quantitative decision making in industry by supporting current trends in software engineering, scalability through cloud infrastructures and multiple programming and modeling languages.

4. Optimization Service Architecture

In this section, a microservice architecture of an optimization service solving an optimization problem is proposed. We refer to the architecture as Optimization as a Service (OaaS). It is derived from the background in Section 2 to apply advantages of modern software engineering principles to mathematical optimization tools. The architecture focuses on small, distributed components that are easy to test, maintain, independently and automatically scalable and manageable by a small team of optimization developers. Thus, optimizations can be delivered to clients (such as web-interfaces for users or other services) faster.

The architecture is based on the following assumptions:

1. Solution method is built on stateless components;
2. Optimizations are executed in an *ad-hoc* way;
3. One specific optimization use-case is approached.

We assume that the solution method is built of stateless components to simplify the architecture and still solve the majority of practical use-cases. Finally, the architecture presented in this section is used for one specific optimization problem in which teams are often structured around.

One optimization service requires the following components, whereas *manager* and *workers* are the central components of the OaaS; *messaging system*, *database* and *object storage* are enabling external infrastructure components:

Manager Provides the external client access to the optimization service. It provides the features presented in the following sections to clients which interact with the service: submitting an optimization, retrieving the state and solution and stopping an optimization.

Worker Each instance is responsible for solving one mathematical optimization at a time. The architecture is agnostic to the solution method (thus it can connect to commercial solvers, use metaheuristics or other optimization webservices, such as Fourer et al. (2010) or Czyzyk et al. (1998)).

Messaging system Queues optimization requests for the workers to allow for asynchronous request processing. Managers add a request to the queue when an optimization is triggered and are immediately responsive again. Workers subscribe to the queue to receive incoming requests for optimizations and work on these potentially long-running tasks.

Database Stores metadata for optimization requests, such as state changes, identifiers or submission information (e.g. username). Solely the manager reads and writes to this database.

Object Storage Stores necessary raw data for the optimization in a price and performance efficient storage system (e.g. an object storage). This includes data provided at the time of request submission, debug or intermediate data of the worker and the final optimization solution. Both the manager and worker read and write from and to it but not concurrently.

In Figure 3 we highlight the interaction of clients with the optimization service. The figures shows two optimization services, A and B, solving different optimization problems. Those services are potentially developed by different teams, in different locations using different program languages. Clients allow users to select data for the optimization, adjust parameters or define an optimization scenario. Often, each client is built specifically for one optimization problem, but the architecture is flexible to cater for clients accessing one or multiple services based on their scope.

Next, we present the different features of the Optimization Service in more detail. The manager's features are similar to the hookup language proposed by Fourer et al. (2010) and we present interactions between components using standardized UML sequence diagrams.

4.1. Submitting a new optimization

The first step of a client it to submit an optimization, see Figure 4. It attaches necessary data as payload to the request. This data can be of various format and size, depending on the use-case. It can be either a reference to existing data (such as a scenario identifier), a user configuration of the optimization, the entire instance for the optimization or a combination of the three. Each option has different advantages and disadvantages and trade-offs, such as size of the payload, availability to the client and accessibility to internal identifiers.

After the manager receives an optimization request, it creates a globally unique identifier and stores it together with metadata in a database. Afterwards, it stores the payload in an object storage, suitable for large raw data. If the two steps succeed, the manager populates a message to the queue indicating the new request can be optimized. The message contains the unique request identifier, so that the worker can refer to this request. Finally, the client receives a success notification together with the identifier of the request. It uses this identifier to refer to the request in other interactions with the Optimization Service.

4.2. Solving a submitted optimization

The core part of the application is the worker, responsible to run optimizations. This long-running component subscribes to a queue to which the manager populates new optimization requests (also called messages). If multiple worker instance are running, the messaging system selects the worker based on a load-balancing mechanism, such as round-robin (see e.g. Shreedhar and Varghese (1996)). When a worker receives a message, steps shown in Figure 5 are executed.

It receives the message and retrieves the corresponding payload from the object storage (using the identifier in the message). This indirect approach has been selected because many message systems impose a message payload size limitation, thus removing the option of sending the entire request payload as a message. The client needs to ensure that the payload is structured in the correct format for the worker. This is part of the contract that the worker defines for its clients. Afterwards, the worker optionally loads additional data for the optimization from other APIs, databases or storage systems.

Usually, the data is transformed for the optimization and solution-method. This can include building the mathematical model in a programming framework, solver specific libraries or data structures used for (meta)heuristics.

Next, the worker starts the solution method (e.g. by calling the solver or starting a heuristic). When it finishes successfully, the solution is stored in the object storage and clients can now download and process the solution.

As shown in Figure 5, workers also communicate with the manager by sending state transitions. We will provide further details on the states in Section 4.6. For now we highlight that the worker indicates when the data loading and preparation is complete and the optimization finished successfully.

4.3. Querying a request's state

After a successful submission, the client polls the manager to check the state of the optimization using the identifier, see Figure 6. The manager reads the latest state of the request from the database and returns it (see Figure 9 for a complete list of possible states), an optional message and the date and time when the state was set.

Clients use different strategies for checking the state of a request: Either long-polling or asynchronously. Long-polling typically blocks the client (e.g. a website) and displays some indication that the optimization is in progress whereas asynchronously refers to the user checking for the update.

4.4. Reading an optimization's solution

When the optimization is solved, i.e. the solution method finished and solution data exists, the client can request the solution of the optimization from the manager, see Figure 7.

First, the manager checks the state of the request by reading it's metadata from the database. If the state indicates that a solution is available (see Section 4.6), the corresponding solution file is read from the object storage and returned (forwarded) to the client. This simple approach makes this functionality independent of the worker's format and data structure. Only the client has to be aware of the interpretation and format of the results.

One can argue the inefficiency of the system by forwarding the data, because the solution is already available to be downloaded from the object storage to safe traffic and processing power in the manager. Indeed, instead of returning the solution, an Uniform Resource Locator (URL) to the raw data in the object storage can be returned. For many use-cases is preferable to secure the access to that URL. This can be achieved by creating a request-specific access-token and returning it together with the URL. Thereby, only the client has the secrets to download and access the solution. Other advantages are faster processing and usually faster download times. However, the authentication for the URL might be cloud-provider dependent and we see it out of scope to evaluate these options. Additionally, clients need to implement another step to download the solution from the returned url.

4.5. Stopping an optimization request

The last feature of the manager is to stop a previously started optimization, see Figure 8. A client requests the manager to stop the optimization by providing the request identifier. The manager loads the request's metadata, checks that the optimization is not already in a terminal state and marks the request as stopped.

One drawback of this simplified approach is that already started workers will continue until they have finished their optimization task. One solution to this problem is using the publish/-subscribe pattern (see e.g. Eugster et al. (2003)): The manager publishes a stop message containing the request identifier to a separate channel or topic of the messaging system. Workers subscribe to that channel and all are notified when a stop message arrives. Each worker checks whether it is currently processing the request with this identifier. In that case, it stops the work and removes the optimization request from the queue by acknowledging the message.

4.6. Optimization request life-cycle

The presented features in this section define a life-cycle for an optimization request, visualized as a state machine in Figure 9. The presented states are a minimum version and more detailed states, in particular for the loading and solution process, can be introduced if necessary for a specific use-case. A request results in one of the terminal states *optimized*, *stopped*, *invalid* or *critical* if at least one worker is available and the solution method is finite.

When an optimization is submitted successfully, it's state is set to *queued* by the manager, indicating that it will be picked up by a worker. Once the request is picked up by a worker, the necessary data is loaded from its sources (e.g. directly from the input data) and prepared for optimization. This can include setting up the mathematical model in a framework or initializing a heuristic. When the problem is ready for the solution method, its state is set to *loaded* by the worker. Once the solution method finishes and the result has been persisted, its state is *optimized* and the manager can return the solution to the client upon request. If the client stops the request, the state is *stopped* and no further state changes are performed.

Exception handling (or error handling) is a critical part of production systems. In scope of the service, several exceptions have to be handled. When the worker recognizes invalid input-data (for instance a wrong format), the state is set to *invalid* to indicate a user-specific error. Expected or unexpected errors can occur while loading data or optimizing a request. Those errors lead to state *error*. Depending on the use-case and/ or the type of error, the system can re-queue the request after some time to retry processing. However, eventually, it transitions into state *critical* if no solution was found after several tries, indicating no further actions will be taken by the service.

5. Use Case: a Bin-Packing Optimization Service

In this section, the benefits of the proposed architecture are discussed in the context of the implementation of a decision support system. The use case is the deployment of a bin-packing solver as a web service.

5.1. The context

In in order to improve the efficiency of internal processes, a company decides to invest in implementing a new decision support system that will be used by teams accross the world to manage daily operations. Today the same operations are performed mainly manually supported by tools with low data integration and poor user experience. Being a new initiative, the company decides to use an agile development approach and leverage modern cloud technology.

Working agile means the core strategy is based on releasing *minimum viable products* (MVP) and incremental updates. That means it is important to engage users early and in real-life conditions in order to provide real value. What does this mean for the design of a decision support engine?

Preliminary analysis indicates the suggestion process can be casted as a bin-packing problem. The exact variant, the usual size and the expected performance is still to be determined. Acknowledging the lack information and thus uncertainty with the final model definition, the best approach is to start from a simple and extensible approach, both in term of functionality and technology.

The development team decides to focus on a general formulation, limiting the amount of details and try not to focus too much on performance.

A concrete implementation of our framework require at this stage the following main decisions:

Model formulation and solution strategy The choice of the solution methods for the bin-packing problem is beyond the scope of this use case and we refer to Ong et al. (1984) for solution methods. For a first version, we have selected a simple approach using mixed-integer programming, see Section 5.2.

Data input/output contract Clients accessing our service must be able to provide the relevant information to the worker in order to setup and solve the problem instances of interest. At the same time the worker must save the solution of each instance in a format that the client can understand and relate to the request input. We provide more insights in Section 5.2.1.

In terms of technical prerequisites, the following steps should be addressed:

Code base It is crucial to store all code and documentation in version control system such as git, mercurial or svn.

Infrastructure Cloud resources such as VMs, message queue and storages must be selected from one or multiple cloud provider. The repeatable creation of those can be scripted in all cloud providers and includes the deployment of the web service. Thus, infrastructure can be created across different environments and for disaster recovery.

CI/CD Setup build and deploy pipelines on a CI/CD system to allow the deployment of both the infrastructure and OaaS.

5.2. The Bin-Packing Problem

The worker implements the bin-packing problem that assigns a set of items to bins with the objective to minimize the number of bins. It was first presented by Johnson et al. (1974) and can be formulated as an integer problem as follows:

$$minimize z = \sum_{i=1}^{n} y_i \tag{1}$$

$$s.t. \sum_{j=1}^{m} w_j x_{ij} \leq c y_i, \qquad i \in N = \{1, \dots, n\} \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j \in M = \{1, \dots, m\} \tag{3}$$

$$y_i \in \{0, 1\}, \qquad i \in N \tag{4}$$

$$x_{ij} \in \{0, 1\}, \qquad i \in N, j \in M \tag{5}$$

Each item j has a weight w_j and each bin i the capacity c (for simplicity, we are solving the homogenous bin-packing problem). The problem specifies n bins and m items. Binary variables y_i indicate whether container i is active and x_{ij} whether item i is assigned to container j. The objective 1 is to minimize the number of bins so that each item is assigned to exactly one bin (Constraints 3) and each bin's capacity is not exceeded, see Constraints 2.

5.2.1. Data Input/Output The input (payload data when submitting an optimization) and output of the worker are assumed to be JavaScript Object Notation (JSON), which is a text-based exchange format, similar to XML. Although more structured formats (such as protocol buffers, see Popić et al. (2016)) can be used, JSON serves as a good showcase for the implementation. Since the required data-set is relatively small and does not depend on external services, we assume the entire instance is sent as payload to the manager. Figure 10 shows the structure of the input data for the bin-packing use-case.

The optimization worker assumes the properties to exist in the request payload. In the payload, $bins$ refers to n, $capacity$ is the capacity c of each bin and $items$ store both a unique identifier and the weight w_j per item. This object-oriented approach is often used to represent structured input data for optimizations.

Similar, the worker produces and stores JSON output that the client downloads through the optimization manager, see Figure 11. In contains the objective value, a list of active bins (it creates bin "objects" based on the number of bins provided in the input, naming them with ascending numbers, starting form 0) and assignments of items (by their unique identifier) to bins. This data structure can easily be presented in a user-interface or further processed in other APIs. In this particular instance, the optimization requires to activate both available bins due to the capacity of 20 and each three items’ weight of 10.

5.2.2. Laying the ground works Preparing for the development and the deplyoment of the system can be done from the very beginning of the project.

1. Establishing code repositories.
2. Agree on coding languages (and their basic guidelines).
3. Setup CI/CD pipeline to build the service.
4. Setup the infrastructure.
5. Setup the deployment.

Some of this steps might come for free after the first time the OaaS framework has been used. Notice that some of the choices about which technology or which tooling might be already dictated by a company policy or agreement with vendors (this is often the case with cloud providers).

5.2.3. Implementation Cycle With the ground work in place, the team can focus on the implementation of the engine. The main goal is to quickly provide a working service to users.

From literature review and previous experience, a simplified model and a general solution approach is selected. For instance model (1) is selected and solved by a general purpose MIP solver. Although not the most efficient approach, it does not require much coding.

Once deployed, the system connects to the application backend and is ready to serve user requests. Early adopters and testers starts to provide feed back and report bugs. The team can improve the service and adapt the functionality to the users requests, until the service is deemd to be ready for a first release. This does not mean the product is completed, but only that can start to give value to the users. New requirements can be considered fast and the feedback cycle repeats.

The proposed OaaS framework allows developers to:

- Deploy new versions (including bug fixes) in small batches and possibly with no down time.
- Leverage managed and reliable infrastructure.
- Collect usage telemetry from users to understand adoption of optimization.

We note that the OaaS framework itself should be subject to refinement along the development process, incorporating learnings to the next iteration.

5.3. Benefits for development and engineering processes

In this section we revise the key benefits from the adoption of OaaS with respect to the challenges as in Section 1 and 2.

5.3.1. Agile development The OaaS allows for quick deployment of a production-like system, and that fits well with an MVP approach, where the focus is on have something up and running quickly and learn from it: (I) the project as a whole benefits from quickly engaging with users with a fully featured proof-of-concept, leading to trust in the product and earlier feedback; (II) the decision support engine collects data (usage patterns, request size, etc..) to shape the next version.

Therefore the technology must also support smooth deployment of new version of a system. The microservice approach isolates the core logic in one place (the worker), so that can be redeployed keeping the rest of the service untouched.

Finally, transparency is not only a by-product of technologies such as version control, infrastructure scripting and CI/CD pipelines but also the consequence of exposing a microservice on the cloud, with other projects potentially taking advantage of the service existence.

5.3.2. DevOps Culture The possibility to script both infrastructure, service deployment and CI/CD pipelines allows clear configuration and traceability, as all scripting can be maintained in source control systems. OaaS is therefore easy to share and configure, as a new service only needs minimal new configuration to start with.

The transparency and the availability of build and deployment automation lead to a general adoption of coding best practices such as unit testing, performance testing and code styling, which raise the overall quality of the code.

5.3.3. Cloud and Microservices The OaaS can be based on mainstream technology for storage, queuing and service orchestration, all available from the large cloud providers. Most of these technologies are moving towards open standards, so that it will be easier to switch from one provider to another.

The manager and worker architecture is a good example of a microservice setup using well established technology (containerisation and orchestration) and supports many use cases. In particular resource requirements such as number of manager/ worker instances, memory and CPU requirements can be adjusted dynamically.

OaaS only exposes the manager to the client, with reduction of security concern. The asynchronous approach reduces the possibility of timeouts on the client requests: Although the solutions methods are fast, network and storage latency might lead to slow request processing.

The availability of a system like OaaS allows to cater for many use cases in a robust and predictable way. The latter point should not be underestimated: OaaS behaves the same way from a client perspective, because all its incarnations share the same manager component. Therefore it is easy for developers to plug and play new OaaS-based services.

6. Conclusion and Outlook

In this paper, we present an architecture to run mathematical optimization in microservice environments, for instance in private and public cloud infrastructures. We focus on running ad-hoc optimizations, initiated by users or another applications. The technology can also be used for slightly simpler batch-optimizations with the main focus on the worker.

After providing technical background, we review the state-of-the-art and derive the scientific contribution. We outline the high-level architecture and present data and information flows between the different optimization service components. Afterwards, an example use case demonstrates implementation details and the general applicability across different programming languages and solution methods.

We conclude that running production optimizations in industry is nowadays connected with additional skillset requirements, such as distributed computing, cloud computing, microservice architectures, DevOps and agile development approaches. With this work we want to highlight the importance of increasing these skillsets for mathematical optimization experts in industry, education and research, to increase practical applicability, preparing for industry requirements and making research available to the public. The proposed architecture presented in this paper eases the applicability of cloud computing to Operations Research and supports focusing on the core aspects of optimization experts: implementing, testing and improving mathematical optimization.

Future work should be done to provide additional common functionality, such as improving the stopping mechanism and suggesting an efficient warm-start strategy but still utilize stateless components (and thus emulate statefulness). Besides, extensions to solve a single request on multiple workers concurrently could be a relevant future extension.

References

Amazon (2019) What are microservices? https://aws.amazon.com/microservices/, [Accessed 14-May-2019].

Beck K, Beedle M, Van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R, et al. (2001) Manifesto for agile software development. URL https://agilemanifesto.org/.

Czyzyk J, Mesnier MP, Moré JJ (1998) The neos server. *IEEE Journal on Computational Science and Engineering* 5(3):68 – 75, URL http://dx.doi.org/10.1109/99.714603.

Di Francesco P, Malavolta I, Lago P (2017) Research on architecting microservices: trends, focus, and potential for industrial adoption. *2017 IEEE International Conference on Software Architecture (ICSA)*, 21–30 (IEEE), URL http://dx.doi.org/10.1109/ICSA.2017.24.

Dmitry N, Manfred SS (2014) On micro-services architecture. *International Journal of Open Information Technologies* 2(9), ISSN 2307-8162.

Docker (2019) Docker. https://www.docker.com/, [Accessed 29-April-2019].

Dunning I, Huchette J, Lubin M (2017) Jump: A modeling language for mathematical optimization. *SIAM Review* 59(2):295–320, URL http://dx.doi.org/10.1137/15M1020575.

Duvall PM, Matyas S, Glover A (2007) *Continuous integration: improving software quality and reducing risk* (Pearson Education), ISBN 9780321336385.

Ebert C, Gallardo G, Hernantes J, Serrano N (2016) Devops. *Ieee Software* 33(3):94–100, URL http://dx.doi.org/10.1109/MS.2016.68.

Eugster PT, Felber PA, Guerraoui R, Kermarrec AM (2003) The many faces of publish/subscribe. *ACM computing surveys (CSUR)* 35(2):114–131, URL http://dx.doi.org/10.1145/857076.857078.

Fourer R, Ma J, Martin K (2010) Optimization services: A framework for distributed optimization. *Operations Research* 58(6):1624–1636, URL http://dx.doi.org/10.1287/opre.1100.0880.

Fowler M (2014) Microservices -a definition of this new architectural term. https://www.martinfowler.com/articles/microservices.html, [Accessed 14-May-2019].

Gray J, Rumpe B (2015) Models for digitalization. *Software & Systems Modeling* 14(4):1319–1320, URL `http://dx.doi.org/10.1007/s10270-015-0494-9`.

Gurobi (2019) Gurobi Cloud. `http://www.gurobi.com/products/gurobi-cloud`, [Accessed 29-April-2019].

Hightower K, Burns B, Beda J (2017) *Kubernetes: Up and Running: Dive Into the Future of Infrastructure* (" O'Reilly Media, Inc."), ISBN 9781491935675.

Hoda R, Salleh N, Grundy J (2018) The rise and evolution of agile software development. *IEEE Software* 35(5):58–63, URL `http://dx.doi.org/10.1109/MS.2018.290111318`.

IBM (2019) Ibm decision optimization on cloud. `https://www.ibm.com/us-en/marketplace/decision-optimization-cloud`, [Accessed 29-May-2019].

Johnson DS, Demers A, Ullman JD, Garey MR, Graham RL (1974) Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on computing* 3(4):299–325, URL `http://dx.doi.org/10.1137/0203025`.

Lukasiewycz M (2008) Java Interface to ILP Solvers. `http://javailp.sourceforge.net/`, [Accessed 29-April-2019].

Mitchell SA (2019) PuLP. `https://pypi.org/project/PuLP/`, [Accessed 29-April-2019].

Ong HL, Magazine MJ, Wee T (1984) Probabilistic analysis of bin packing heuristics. *Operations Research* 32(5):983–998, URL `http://dx.doi.org/10.1287/opre.32.5.983`.

Popić S, Pezer D, Mrazovac B, Teslić N (2016) Performance evaluation of using protocol buffers in the internet of things communication. *2016 International Conference on Smart Systems and Technologies (SST)*, 261–265 (IEEE), URL `http://dx.doi.org/10.1109/SST.2016.7765670`.

Red Hat (2019) rkt. `https://coreos.com/rkt/`, [Accessed 29-April-2019].

Schwaber K (2004) *Agile project management with Scrum* (Microsoft press), ISBN 9780735619937.

Shahin M, Babar MA, Zhu L (2017) Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access* 5:3909–3943, URL `http://dx.doi.org/10.1109/ACCESS.2017.2685629`.

Sheikhalishahi M, Laganà D, Grandinetti L (2012) Operations research as a service. *CLOSER*, 480–483.

Shreedhar M, Varghese G (1996) Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on networking* 4(3):375–385, URL http://dx.doi.org/10.1109/90.502236.

Smirnov S, Voloshinov V, Sukhoroslov O (2016) Distributed optimization on the base of ampl modeling language and everest platform. *Procedia Computer Science* 101:313–322, URL http://dx.doi.org/10.1016/j.procs.2016.11.037.

Steglich M (2016) Cmplserver-an open source approach for distributed and grid optimisation. *Anwendungen und Konzepte der Wirtschaftsinformatik* ISSN 2296-4592.

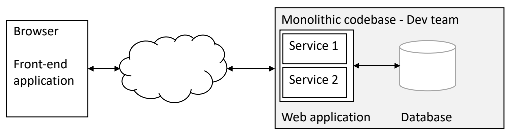
Stolberg S (2009) Enabling agile testing through continuous integration. *2009 Agile Conference*, 369–374 (IEEE), URL http://dx.doi.org/10.1109/AGILE.2009.16.

Ustundag A, Cevikcan E (2017) *Industry 4.0: managing the digital transformation* (Springer), URL http://dx.doi.org/10.1007/978-3-319-57870-5.

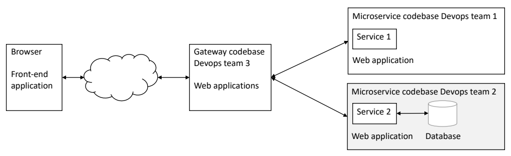
VersionOne C (2019) 13th annual state of agile report. https://www.stateofagile.com, [Accessed 03-June-2019].

Villamizar M, Garcés O, Castro H, Verano M, Salamanca L, Casallas R, Gil S (2015) Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. *2015 10th Computing Colombian Conference (10CCC)*, 583–590 (IEEE), URL http://dx.doi.org/10.1109/ColumbianCC.2015.7333476.

Virmani M (2015) Understanding devops & bridging the gap from continuous integration to continuous delivery. *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, 78–82 (IEEE), URL http://dx.doi.org/10.1109/INTECH.2015.7173368.

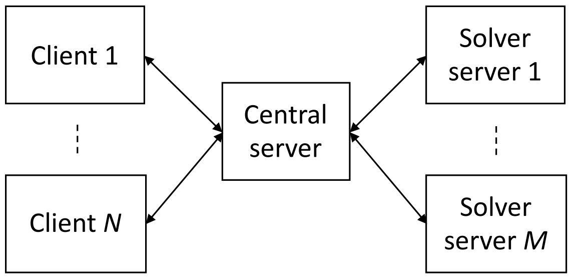


(a) Monolithic architecture.

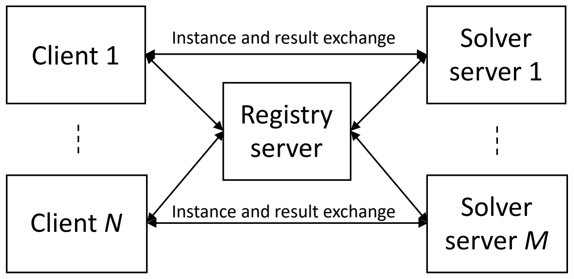

Browser
Front-end application
Gateway codebase Devops team 3
Web applications
Microservice codebase Devops team 1
Service 1
Web application
Microservice codebase Devops team 2
Service 2
Web application
Database

(b) Microservice architecture.

Figure 1 Monolithic vs. microservice architectures (Villamizar et al. (2015)).



(a) Centralized computing architecture as in Czyzyk et al. (1998).



(b) Service-oriented computing architecture as in Fourer et al. (2010).

Figure 2 Different computing architectures (Fourer et al. (2010)).

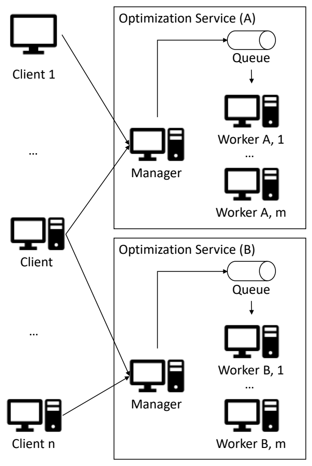


Figure 3 Overview of the optimziation service architecture.

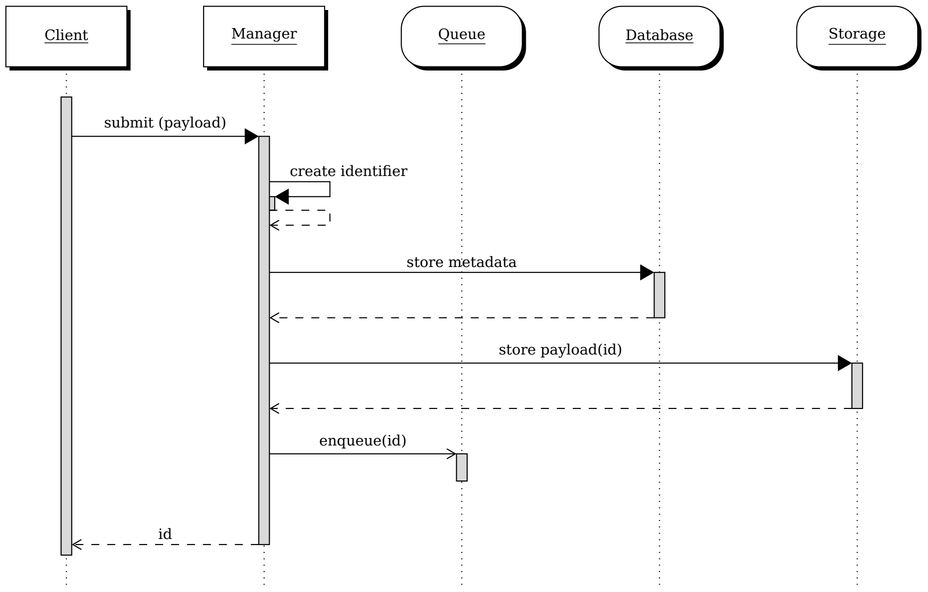


Figure 4 Optimization Service UML sequence diagram to submit an optimization.

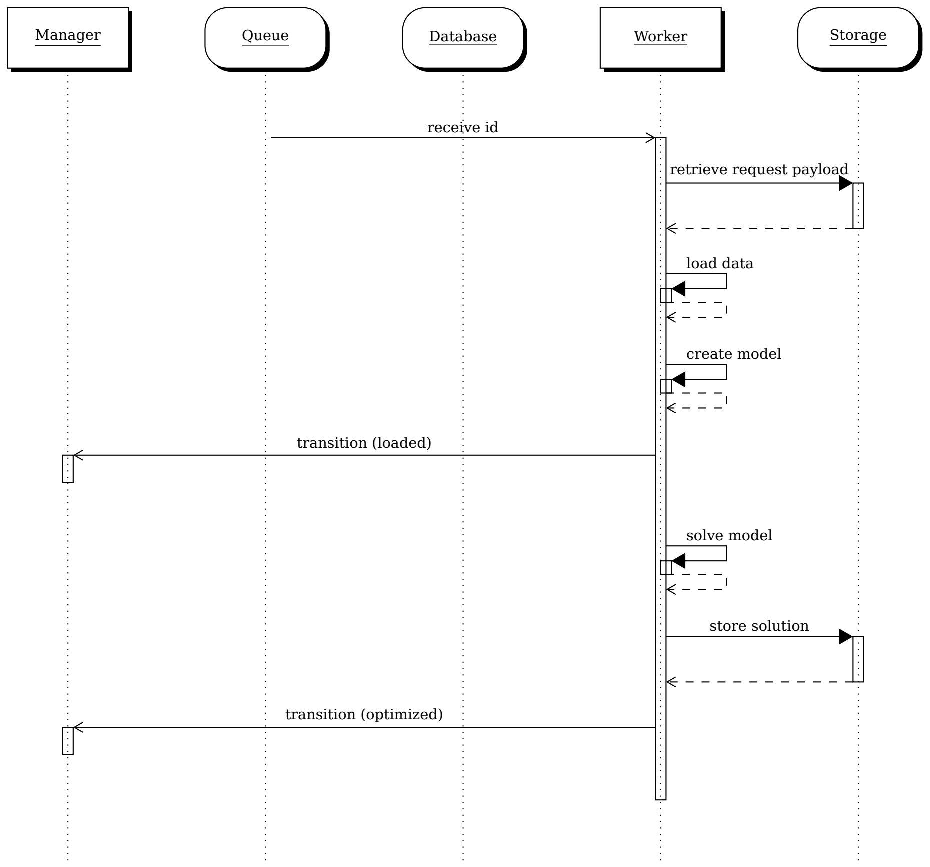


Figure 5 Optimization Service UML sequence diagram to solve an optimization.

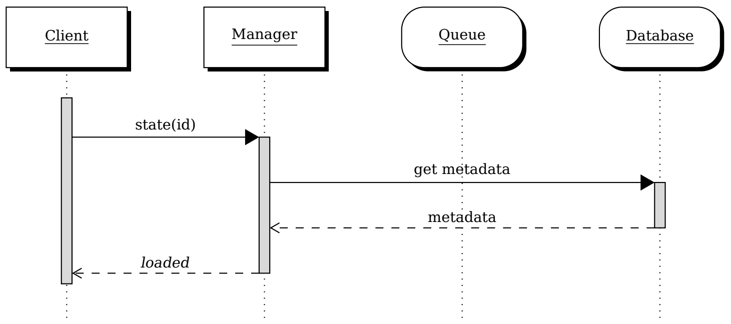


Figure 6 Optimization Service UML sequence diagram to retrieve the optimization state.

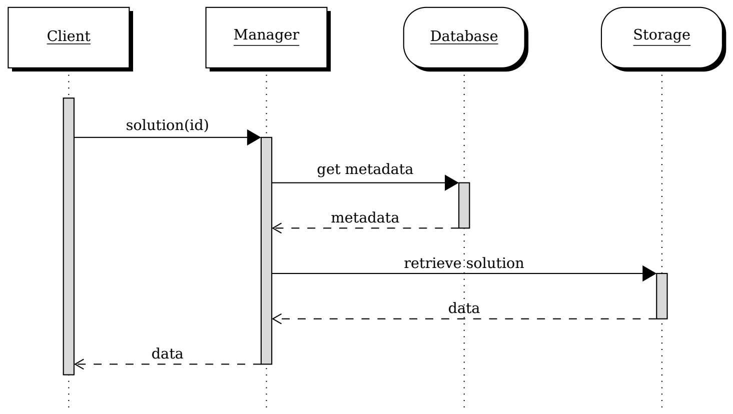


Figure 7 Optimization Service UML sequence diagram to retrieve optimization solution.

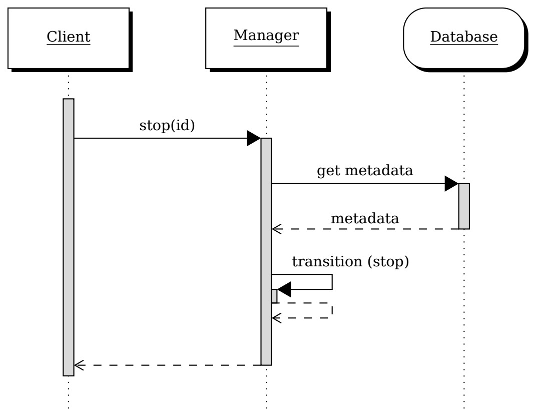


Figure 8 Optimization Service UML sequence diagram to stop an optimization.

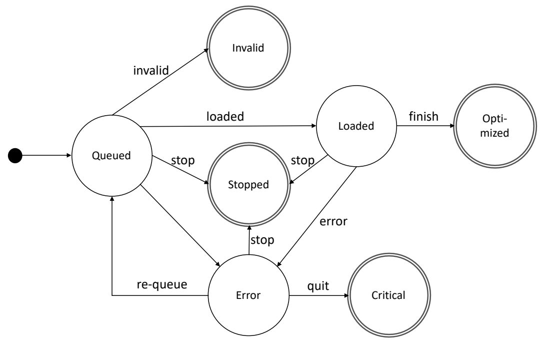


Figure 9 State machine showing optimization request states and transitions.

```
{
    "bins": 2,
    "capacity": 20,
    "items": [
        {
            "id": 1,
            "weight": 10
        },
        {
            "id": 2,
            "weight": 10
        },
        {
            "id": 3,
            "weight": 10
        }
    ]
}
```

Figure 10 JSON structure of the input data for the bin-packing use-case.

```
{
    "objective": 2.0,
    "active_bins": [0, 1],
    "assignments": {
        "3": 0,
        "1": 1,
        "2": 1
    }
}
```

Figure 11 JSON structure of the output data (solution) for the bin-packing use-case.