ENGEN582-24X
Honours Research and Development Project

Integration of cloud computing paradigms for performant analysis of simulated process engineering applications.

Caleb Archer

Supervised by Tim Walmsley, Mark Apperley

Contents

1. Introduction

The world of process systems engineering has seen substantial growth since its burgeoning in the 1950s [1], creating and serving critical tools for chemical engineers working in industry and academia alike for improving the performance of resource-intensive processes. Process modelling, simulation and optimisation software solutions have played a substantial part in this journey. Over the same period, the nature of software engineering has evolved rapidly, with new paradigms swiftly overtaking their forebears, and pushing forward our understanding of what software can accomplish.

One such paradigm, cloud computing, has positioned itself as critical to the efficient and effective operation of businesses around the world. It is the goal of this literature review to explore how cloud computing oriented software architectures could be better aligned with the process engineering software space to improve upon the performance, capability, and accessibility of such software solutions. Specifically, the research questions to be answered by this review are:

- What architectures are available for deploying and running systems on cloud environments?
- What challenges and limitations are being faced by process simulation software?
- What tools or techniques have been used for assessing the performance of different software architectures?

Firstly, this review explores the space of software architecture, delving into the ways in which software systems are designed and arranged, how they have evolved, and what purpose each of these serves. Next, the state of process engineering software will be assessed, including interactive (user-interface based) and programmatic (computer-based mathematical modelling) approaches to performing simulation, modelling, and optimisation of chemical processes. The purported benefits and drawbacks of each approach will be discussed, as well as any wider issues with the area overall. Following this, we will determine how self-adaptive computing has influenced modern distributed computing, and what systems are actively being used in this realm. Finally, the field of software performance modelling and testing is be evaluated, focusing on how the associated techniques can be used to assess the performance of implemented software architectures.

2. Software deployment architectures

For various reasons, new software architectures evolve out of others, often substantially changing the nature and behaviour of the underlying systems, as well as the problems faced by engineers working with said architectures. The appropriateness of a given architecture can be considered dependent on the needs of the business or entity deciding on it.

2.1 Evolution of modular software systems

Parnas, in his seminal 1972 paper, described modularity as a means of improving the coherence and future changeability of a system [2]. Systems should be able to be decomposed into modules such that teams can independently contribute to the system without running into conflicts. These modules should have little knowledge of how other modules have been designed, and usage of other modules should be easily interchangeable. Applying these ideas, as described by Parnas, should lead to less engineering effort over time, ease of refactoring the target system and ease of understanding the system. The examples of applying modular design to a system focused on standalone programs, where one module is one isolated section of code. Despite this contextual view, these principles can and have been applied to systems since the publishing of this paper.

Parnas *et al.* clarified these principles of modularity in a later 1985 paper [3], after identifying a gap in the industrial application of software engineering practices versus those discussed in academic contexts. They took a non-trivial software system that already existed and redesigned it using modularity principles in order to compare the result to the original, rather than justifying the principles via small-scale, toy examples. Through their exploration, they emphasised the specific idea of information hiding, as well as the use of specific module creation guides for projects to help engineers to use information hiding within the context of a specific project (or sub-module). Though this paper also focused on standalone (but internally decomposed) software systems (as in [2]), it showcased the practical importance of constructing a system in a layered fashion, with thin connections between those layers. McGregor evaluated the modularity that can be achieved with the use of the model-view-controller design pattern, which separates program logic into modules that handle system interaction, data interaction, and the space in between the two. They emphasised the use of diagramming and related techniques to determine the breakdown of a system prior to beginning development [4].

The previous papers, however, spent little time exploring how a system decomposed into modules could contribute to reusability within the system, which was addressed by Almeida *et al.* [5]. To promote repeated usage of system components and modules, they constructed a guiding framework for reusability called RISE (Reuse in Software Engineering). This framework sought to define the different aspects of reuse, not only from a technical standpoint, but also an organisational one, including the reuse of knowledge and processes. Application of these ideas was purported to speed up the overarching engineering process, as described by Parnas [2] and further evidenced by Parnas *et al.* in 1985 [3].

Sifakis explored in more detail the interaction of software components across potentially distributed systems in [6], defining a formal method for verifying the properties and behaviours of such interactions. Though it approached system composability from a formal, mathematical view, it contributed to expanding the domain of defining modular systems from a standalone software system to a distributed one.

As an example of a more concrete step towards distributed modularisation of systems, Goncalves *et al.* depicted the process, and related impacts, of refactoring a monolithic software system into a more modular one with clear domain separation, which were all originally addressed by the monolith [7]. However, the study completed only represented the intermediary step of standalone modularisation,

without expanding towards the ultimate stated goal of moving towards microservices.

2.2 Evolution of parallel computation

Parallel computing involves the simultaneous processing of a defined algorithm, such as simultaneously calculating the gradients for a set of differentiable equations, as opposed to calculating them one-by-one, which would be defined as sequential computing.

A 1982 survey by Haynes *et al.* was conducted regarding the state of parallel computing at the time [8]. It showed that parallel systems were, for the most part, designed with esoteric chip architectures to maximise the amount of parallelism that could be achieved. Along with this, to handle large-class problems that a single parallel machine could not solve on its own, a large number of these machines were networked together. Such systems were considered to have grown beyond the traditional single-CPU system capable only of attending to tasks in a serialised fashion. In progressing to 2006, this view of parallel computing became multi-faceted, with Asanovic *et al.* addressing the arrival of multi-core computation in retail hardware [9]. Their view was that multi-core systems would face similar challenges to traditional parallel computing systems, and that instead small-scale "manycore" systems would be a better target, though this has not eventuated in the context of modern CPU architectures.

In response to the aforementioned growing presence of multi-core retail computer systems, a study towards converting existing sequential programs to partially parallelised versions [10] was performed in 2007. Bridges *et al.* argued that the arrival of affordable multi-core systems did not come with a similarly increasing utilisation of these additional cores, attributed to the increased complexity involved in manually defining parallel processes. A framework was used in demonstration of thread extraction of a benchmarking suite, increasing the performance of this benchmarking software by several times. Asanovic *et al.*, in a later 2009 paper [11], continued to push for adoption of "manycore"-centric programming models, stating that a large bottleneck in reaching full exploitation of additional cores on microprocessors is the programmer, who is not typically acquainted with writing programs in a parallel fashion. Their proposal, similar to Bridges *et al.*, was to construct an autotuner for introducing parallelism to existing sequential software. However, as shown in [12], some programs are easier to parallelise than others, and may not see the expected gains as shown in examples of this particular paper.

Chen *et al.* provided a survey on parallel computing [13], its architectures, methods and algorithms, performance, and its applications. At a high level, there are two types of parallel computing architectures: the Common Parallel Computer (CPC), and the Supercomputer. CPCs are relatively cheap, commonly available computers that can be found in the home, at workstations, and in pockets, with multi-core CPUs. Supercomputers, on the other hand, are far less accessible, but still demonstrate far greater computing power capability for massively parallel tasks. This may be a simplistic model, especially with the entrance of the GPU (Graphics Processing Unit), as discussed by Xu *et al.*, where they focused not on the architecture of parallel computing, but general styles (or scales) in which many different architectures can fit [14]. These styles include internet, datacentre, process and accelerator scales of parallel computing. The GPU may be considered an alternative actualisation of general "manycore" computing as introduced previously by Asanovic *et al.*

The refactoring of an electric-market simulation software towards multi-core parallelisation [15], and subsequent results, was shown by Seveso *et al.* The application was originally written and run in a sequential fashion, but faced performance limitations as its usage was expanded for simulation scenarios of greater complexity, where the time taken to undertake a simulation exceeded acceptable runtime boundaries. Substantial improvements to some aspects of the simulation software were seen, but the authors neglected to provide a full analysis of simulation time between the sequential and parallel ar-

chitectures.

Vivas *et al.* provided a better characterisation [16] of performance differences between different architectures. Both HPC (High-Performance Computing) and non-HPC styles of parallel computing are available as choices for researchers conducting scientific computing. Non-HPC systems can be made up of existing computing hardware available within a research institution, serving as an alternative to purchasing dedicated supercomputing hardware. Though it was found that dedicated HPC hardware performed well beyond each of the networked clusters, it would have been beneficial to provide a quantitative assessment of reliability across these different architectures.

With parallel computing now more accessible by default via multi-core retail computing systems, as well as relatively inexpensive dedicated parallel processors (such as the GPU), it has become substantially more viable for software systems built for general hardware to take advantage of parallelism.

2.3 Evolution of distributed systems

Distributed systems are software systems made up of a network of potentially heterogeneous computing hardware (nodes), where the tasks running on each node may not all be the same as every other node. The rise of networked devices has enabled software systems to move beyond the format of a standalone application running locally on one machine. Such distributed systems may even be geographically heterogeneous, as is seen with cloud computing. This paradigm can be viewed as an evolution of both parallel computing and modular software design, attempting to take advantage of additional computing hardware in order to construct more powerful, sophisticated systems. Initially, these systems would have entered the computing domain with each software service requiring an entire machine with one operating system to run securely, but new technologies arose to handle computing resources efficiently.

2.3.1 Virtual machines

Virtual machines are entirely virtualised computers with virtual computing resources, including CPU, memory and storage, assigned to them [17]. A single physical computer can often run several virtual machines, where software running inside the virtual machine has no knowledge of software running in others.

As indicated by Goldberg in their 1974 survey of virtual machine research [17], virtual machines were originally developed to handle the limitations of a rigid kernel interface, where it was difficult to test the behaviour of kernel-level software, or entire operating systems, which require direct access to all functions of the underlying system. Virtual machines enabled this direct access without having to repeatedly move between compatible systems; an entire operating system could be simulated via another base system, and with full resource isolation (CPU, memory, etc.). An early issue identified with virtual machines is the performance overhead, a natural consequence of having two or more kernel layers involved in running virtualised systems.

As time progressed, virtual machines saw their use expanded beyond mere simulation, compatibility and testing purposes. Whitaker *et al.* detailed a relatively new use-case: distributed networked applications [18]. They argued for the usage of lightweight virtual machines to make a service available to a network, which they considered to be more viable than allocating dedicated physical machines to each service (which may require resource isolation for security and performance purposes). With individual computers being capable of running multiple virtual machines, they saw this novel multi-tenancy service model as being newly reasonable. However, performance and storage overheads from virtual machines may be a largely unavoidable issue. Li *et al.* also identified performance overheads

as a prominent issue with virtual machines that has seen a great deal of attention within research [19]. As well as this, there has been great emphasis on migration of virtual machine images and resource sharing.

With the advent of cloud computing, virtual machines have played a significant role in providing infrastructure as a service [20], where individuals and organisations can rent virtual machines with defined CPU, memory and storage resources from cloud providers. Hardware may have previously been accessible in dedicated form, typically on-premise, but cloud computing has seen this shift to a distributed form of hardware access. As described by Sharma *et al.*, virtual machines enabled the construction of virtual computing clusters, where seemingly independent machines can be virtually networked together to serve some collective goal. The hard resource limits on virtual machines prove to be problematic when resource efficiency is a desired system attribute, which was previously discussed in [19].

Pietri *et al.* addressed this resource allocation issue directly, discussing research focused on mapping virtual machines to physical hardware [21], especially how virtual machines can be dynamically resized to serve variable application resource requirements. It was pointed out that part of the difficulty of this problem is the computation required to perform any reallocation, requiring carefully designed algorithms for deciding when reallocation should occur. Xiao *et al.* presented such a dynamic allocation strategy in [22], using predictions of future resource requirements to decide when additional virtual machine instances are needed. Conversely, a study of virtualisation history by Randal found that even with these hard resource limits said to provide system isolation, virtual machines cannot be considered equally as secure as systems deployed on isolated hardware, given their proven susceptibility to recent memory-based weaknesses [23]. Despite this, virtual machines continue to serve as foundational tools for cloud computing services.

Virtual machines improved the scene for software deployment substantially, allowing for services to be run on operating systems that would otherwise be incompatible, and paved the way towards cloud computing via secure multi-tenancy. Though they serve as a critical stepping stone towards efficient resource utilisation, their typically large nature prevents systems from taking full advantage of the underlying hardware.

2.3.2 Containers

Containers have emerged as popular alternatives to full virtualisation as seen with virtual machines, and instead rely on strong filesystem access control and process separation to provide a facade of full environment isolation [23]. Their more lightweight nature has seen a presence takeover of containers for direct software deployment purposes [20], where previously an entire virtual machine may have been dedicated to a single software service [18]. Instead of each service being bundled with an entire operating system, as with virtual machines, containers allow for a shared operating system to be used under-the-hood, while separating the required user-level libraries and dependencies into isolated filesystems (see Figure 1).

The utility of distributed cloud computing for bioinformatics was addressed in [24]. The need to share information between biomedical institutes and centres has seen the resource sharing and centralisation benefits of distributed cloud computing receive attention, where the traditional approach has been isolated self-management of required computing infrastructure. Provisioning of computing resources, whether via dedicated hardware, virtual machines or containerisation, is now a trivial task. At the same time, some risks of a move to cloud computing were acknowledged, including data security and dependence on third-parties to deal with threats to infrastructure.

Pahl *et al.* assessed the state of containerisation research and current focuses and prominent use-cases

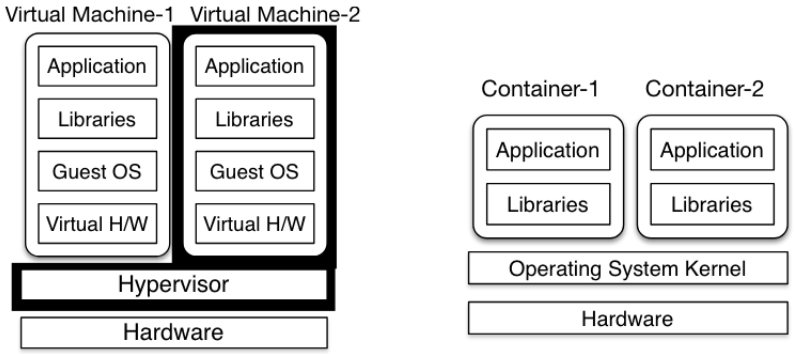

Figure 1: Virtual machine and container stacked architectures [20]

[25]. It was determined that containerisation is heavily present in the software deployment space, being used to package applications to serve to end-users, and being done so across different technical fields of interest. Along with this, they also found a general immaturity of automation of containerised system management, but container orchestration is increasing in usage in order to lower resource usage costs and improve system performance. There seems to be more focus on the flexibility of containerised systems rather than performance, though these two elements can be interlinked. As steps towards improving container management automation, several heuristic techniques for container scheduling and allocation are described in [26], though by nature, these approaches may not be globally optimal, and require consideration of context to drive a system towards superior performance.

The resource utilisation and management benefits of containers were surveyed and explored in [27], within the context of edge and fog computing, rather than just traditional cloud environments. They stated that virtual machines have recently dominated the computing resource allocation domain, but containerisation continues to expand its presence, especially with the substantially lower overhead required to provision a set of containers compared to one of virtual machines. However, a known issue is that many containerised systems are themselves deployed within virtual machines, creating potentially significant overheads for container to host communication, though this is complemented by taking advantage of resource isolation provided by the host virtual machine [20]. As well as this, containers typically do not adhere strictly to resource usage limits defined by runtime managers, in opposition to standard virtual machine behaviour. Vaño *et al.* elaborated on the use of containerisation technology [28], outlining some elements of public cloud computing that have been adopted in edge computing. While several container management technologies at the edge look promising, such as K3s or MicroK8s, they point out that problematic attributes of containers, including image size storage requirements and minimal application isolation, serve as challenges to adoption at the edge.

Containers are powerful enablers of efficient resource utilisation without any redundant performance and storage overheads. Subsequently, this nature has allowed for distributed software systems to consist of machines running many more services than before, while retaining clean environment separation.

3. Process systems engineering software

At a high-level, there are two general approaches to performing process modelling, simulation and optimisation. The most common approach used in industry is interactive process modelling, involving locally available GUI (Graphical User Interface) software. An approach with more direct utilisation in academia is mathematical modelling, where a model is carefully defined in code.

3.1 Interactive process modelling

Process simulation and optimisation software is heavily used on an interactive basis to enable easy access to modelling of processing plants [29], with tools such as Aspen Plus and Aspen HYSYS serving as common examples of these interactive environments. However, these process modelling tools often lack the models and solvers required to handle some simulation, optimisation or synthesis problems. Such problems need to be solved using less intuitive tools such as Python or MATLAB, or via modelling languages like Modelica and GAMS. Most commercial process modelling tools, including Aspen HYSYS, Aspen Plus, and ProSIM, make extensive use of sequential-modular flowsheet solving [30]. These tools are typically locally run on machines of end-users.

The use of such process modelling software has been validated in both research and industrial contexts [31], where experimental data is often compared with model outputs to determine the level of agreeance. Models found to be accurate can subsequently be used to experiment with different process parameters and configurations and observe downstream and high-level impacts [32]. While not perfect representations of the underlying process, analysis via model usage can be performed quicker than that of experimental analysis, which would also involve risky live changes to plant operations. As depicted in [33], interactive process modelling tools can ultimately be used in an economic optimisation sense, such as identifying points in a process where resource usage can be increased or reduced to improve system efficiency and reduce operating costs.

Klatt *et al.* indicated the importance of process modelling, simulation and optimisation tools and frameworks being easily accessible to non-experts (working in the process industry) within interactive packages [1]. The process engineering industry is heavily dependent on these tools, in interactive form, to apply modelling and optimisation techniques to their specific contexts. The needs of these industrial practitioners, however, differ from those of researchers. The time required to perform computational optimisation [34] of a process, for example, is a deciding factor for whether certain models or tools are utilised in practice.

3.1.1 Sequential modular simulation

Sequential modular simulation is the flowsheet solving and optimisation method most commonly used by commercial process simulation software [35]. It solves different parts of a process flowsheet in a serial fashion, processing each "module" without consideration for the entire system.

A 1993 study by Harrison [12] of the parallelisation of a sequential-modular simulation algorithm on a supercomputer found that, although some reduction of computation time can be achieved, these improvements are "modest" at best. More complex models were better improved by parallelisation techniques, but the differences were not magnitiudinal. The relatively early nature of this study means modern improvements to sequential modular algorithms and increased uptake of multi-core workstation computers could not be reflected in the improvements shown, but the findings still have implications for sequential-modular simulation today. A similar conclusion is provided by Ralphs *et al.* in their overview of the parallel solving space for MILP problems (mixed integer linear programming) [35]. They state that further magnitudinal improvements to sequential MILP solving algorithms are

highly unlikely, making room for further emphasis on parallelised approaches, which have otherwise been stunted by the emphasis on said sequential solving improvements. Exploration of parallel MILP solving methods is challenging, as existing sequential solvers behave in ways that are fundamentally incompatible or problematic for effectively carrying out parallelisation.

Further exploration of sequential-modular simulation is shown by Kulikov *et al.* [36], where it is applied to the solving of particulate process flowsheets. Standardised variants of this algorithm have traditionally been applied to fluid operations. An process simulation tool called CHEOPS is applied to make use of solvers from various existing toolkits. Their results demonstrated sequential-modular simulation could be applied via existing products and libraries in achieving a fine-grained model result for non-standard process problems. However, this approach, depending on the problem, may require access to several commercial tools with prohibitive licensing costs. The authors also identified that the required computation time, despite efforts to reduce this, is still a problem, due to the underlying iterative nature of sequential-modular simulation. They posit that algorithmic improvements can improve simulation efficiency, though, as identified previously by Harrison [12], a high level of improvement may be doubtful.

A performance assessment of a parallel-modular optimisation algorithm was conducted in [37], an adaptation of existing sequential solving algorithms. Message-passing was utilised across a distributed set of computation nodes, which received instructions via MIMD (multiple instruction, multiple data). A significant performance improvement proportional to the number of nodes used was observed, though little improvement was seen beyond 14 nodes. It may be difficult to generalise this solution, with customised usage of parallel programming libraries required, and the iterative nature of the underlying sequential optimisation still serves as a limiting factor.

While serving as the core driver of most interactive flowsheet optimisation, sequential modular simulation is seen as having little opportunity for performance improvements, and cannot be used for solving certain optimisation problems.

3.2 Mathematical process modelling

With improvements to sequential-modular solving and optimisation being few and far between [35], other mathematical modelling approaches are starting to present themselves as more viable tools for industrial use [38]. An increase in general available computing power has made previously intractable (or otherwise prohibitively expensive) problem formulations possible to solve, especially with respect to Mixed Integer Non-linear Programming problem (MINLP) formulations. However, global optimisation of processes still presents a challenge for researchers, and so the field has continued to shape leading up to wider industrial application. These direct mathematical formulations require deep understanding of each underlying problem and the techniques needed to correctly formulate said problem [39], making generally accessible mathematical modelling approaches generally inaccessible to non-expert users. De Tommaso *et al.* affirmed the "solid theoretical background" required to perform process simulation, but ignored the significant industrial audience these tools have [40]. Even with a general modelling approach defined, the requirement to have good understanding of computer programming and mathematical modelling proves problematic to wider usage of these techniques. Martín *et al.* additionally identified the issue of open access to models [39]. Advances in process modelling research have not been evenly accompanied by availability of the newly-created or modified models, particularly with respect to sharing code.

Increases in computer processing power have triggered further development of mathematical modelling techniques such as superstructure optimisation [41]. As compared to sequential solving methods (such as sequential-modular simulation), superstructure optimisation attends to the entire flowsheet simulta-

neously. The approach struggles to reach industrial usage due to prior knowledge needed to initialise the model, as well as computational difficulty. Such a method for performing process systems synthesis was described by Duran *et al.*, involving a Mixed-Integer Nonlinear Programming (MINLP) formulation [42]. A superstructure master problem is defined, where subsequently derived sub-problems are solved recursively. This algorithm, while somewhat generalisable, only consistently applies to convex functions (such as a function of a quadratic shape).

According to Klatt *et al.*, early efforts towards mathematical modelling within process engineering were conducted in an exploration of how the computer could help solve substantially sized flowsheet models [1]. This research was performed as a response to otherwise ad-hoc design thinking that was often applied to engineering process systems at the time. Despite a long history of research, these high-performance models are sparsely applied in industry. Such models could be utilised for real-time optimisation, but once again, find themselves underrepresented in practical usage. For greater prominence of these numerical methods to arise, they would need to be made available to non-expert users via existing or new interactive process modelling software.

Nayak *et al.* described the creation and design of an open-source, equation-oriented process modelling framework [43] called OpenModelica, which uses a process modelling language named Modelica. It consists of a pre-defined set of model components and functions which can be re-combined together to construct context-specific models. Though it may be very useful for educational and research purposes, the framework does not address the issue of the skill required to utilise modelling frameworks and libraries in general, and is unlikely to see industrial applications.

3.2.1 Equation oriented simulation

Equation oriented simulation is commonly placed in opposition to sequential modular simulation, as it is encompasses simultaneous solving and optimisation approaches, though it is less available in interactive form. Within a broad overview of Process Systems Engineering history, [44] indicates that equation-oriented simulation has widened the scope for optimisation of process engineering activities, and beyond, as compared to its sequential-modular counterpart.

Biegler continued this thought [30], saying that despite the overwhelming popularity of sequential-modular flowsheet solvers, they are overly sensitive to convergence failure due to dependence on gradient approximations, and therefore limit the subset of optimisation problems that can be solved. Equation-oriented solvers, on the other hand, create such a speed-up that they can be used for real-time optimisation (RTO). Biegler stated that equation-oriented simulation and optimisation was not common in commercial flowsheeting software, in part because of an unfilled need for accurate first and second-order gradients.

In [45], in an attempt to improve the usage of equation-oriented optimisation for solving large, complex flowsheets, made use of automatic differentiation to obtain precise first and second derivatives that are needed to ensure model convergence. The experiment was performed on an Intel-based computer, on which they reported an average of 15 CPU-minutes of computation time. While they achieved results consistent with an existing Aspen Plus optimiser, no comparison of the time complexity involved in the calculations was provided. Generalisable GPU-based (Graphics Processing Unit) equation oriented optimisation was tested in [46]. They found that GPU usage could offer performance speed-ups when compute utilisation is high, but smaller flowsheet models performed worse than their traditionally CPU-calculated counterparts due to lower compute utilisation and model complexity. Their experiment involved the testing of one model at a time, but did not consider the possibility of testing individual models simultaneously.

DAE Tools, an equation oriented process modelling framework for Python was proposed in [47]. It provides general capabilities for solving model-based mathematical simulation and optimisation problems, which can then be made embedded as part of offline and online user applications. Although it supports parallel computation, this is only in the form of generated code to be separately run on tightly-coupled parallel systems involving several nodes.

With the goal of attaining performant global optimisation, [48] described a hybrid approach to solving a flowsheets for phase equilibrium models, involving a combination of sequential-modular solving and equation-oriented solving methods. The experiment made use of in-house optimisation tools. The approach depicted was successful in meeting requirements, but requires good problem-specific model formulation and niche knowledge requirements in order for the approach to be usable in other contexts.

The work in [49] applied pseudo-transient (PT) equation-oriented optimisation to flowsheets involving dividing-wall distillation columns. Prior attempts at using equation-oriented optimisation suffered from the need to initialise models with good initial starting points to allow for convergence, considered to be a difficult task. Use of pseudo-transient modelling allows for the range of viable initial starting points to be expanded. The study found that this overall method enables consistent, rapid model convergence of the entire flowsheet for a model, and better enable cost reductions for a given process design problem. A review of PT modelling advances in equation-oriented simulation [50] showcased at a broader level the impact that PT approaches have had on viable and efficient optimisation. PT models can better provide exact solutions for a given optimisation problem, and reduce the work required to initialise a model. Despite this progress, challenges remain for making PT modelling widely feasible. The review outlined that PT modelling has not yet made its way into commercial simulation software due to the difficulty of implementation, and work is ongoing to make PT approaches more computationally feasible. It can be said that, while there has been a great deal of progress made in making equation-oriented simulation a true competitor to the status quo, the lag between research and implementation makes accessibility to this progress challenging.

What can be observed for interactive and mathematical modelling approaches is that there is a performance and accessibility trade-off. Equation-oriented simulation, and general mathematical modelling approaches, are much more performant and reliable modelling and optimisation tools compared to those commonly used in interactive process engineering software. However, they are generally far less accessible to industrial users because of their required expertise and deep foreknowledge of how the underlying models work. They are sparsely available in interactive form, which is, in part, due to ongoing research to make them more generalisable. The interactive approaches to process modelling, which are used far more extensively in industry, suffer from over-reliance on inherently performance-limited solvers, preventing the industry from taking advantage of accurate optimisation at speed. A wider problem with process simulation software is that, even if more performant solving methods are available, these are limited by the localised nature of the desktop software operating environments.

4. Self-adaptive approaches to distributed computing systems

Self-adaptive computing, which addresses self-regulating systems, is a field that has received implicit attention from distributed computing technologies. The advent of microservices and their common modes of implementation have caused this merging of paradigms.

4.1 Foundations of self-adaptive systems

Inspired by the many subconscious and self-regulating functions of the human body, Kephart *et al.* described autonomic computing, an aspirational paradigm with the goals of having computer systems perform self-management, self-configuration, self-optimisation, self-healing, and self-protection [51]. This was in response to the increasing complexity of software systems and the effort involved in maintaining them. The authors acknowledged that this high-level paradigm depiction demands a great deal from engineers and researchers, and also exacerbates the consequence of human-caused error, as errors would instead be made at the goal specification level, having effects on the entire self-managing system.

Weyns provided a more concrete realisation and evolution of the paradigm described by Kephart *et al.*, with a presentation of the history and current state of self-adaptive systems [52]. Weyns affirmed that the complexity involved in managing modern software systems has increased substantially, and that there is a need for such systems to be able to manage and configure themselves autonomously, thus the entrance of the term, *self-adaptive systems*. Over time, self-adaptive system models have evolved from task automation to a much more involved management system responsible for a lower-level sub-system. The conceptual model (as seen in Figure 2) described consists of four components: the environment, the managed system, the defined adaptation goals and the managing system. The most recent iteration of self-adaptive systems has been the implementation of control theory inspired systems, where a feedback-loop drives a managed and monitored system towards its specified goals. Weyns notes that a challenge for this field of research is the empirical validation of the proposed models and strategies, as well as whether the latest wave of self-adaptive systems (rooted in control theory) can be demonstrated as a good foundation for self-adaptive systems in practice.

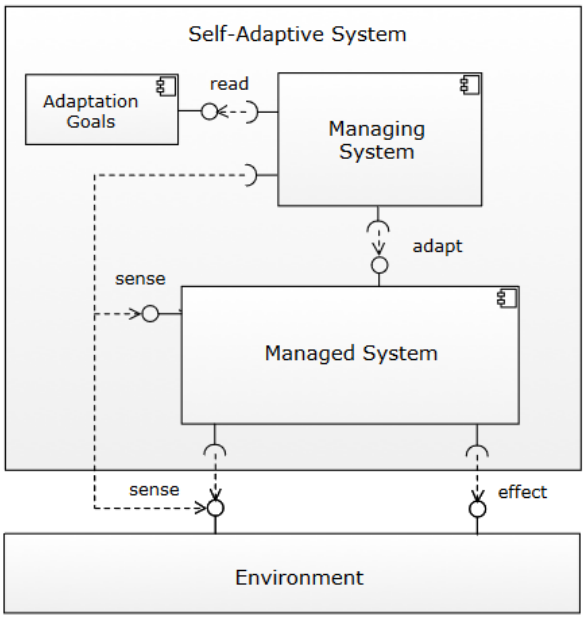

Figure 2: Conceptual model of a self-adaptive system by Weyns [52]

These ideas, while not firm in how they are to be implemented or the scope of implementation, provide a basis for systems to be designed from the very beginning with consideration for reliable performance, with the expectation that they can handle interruptions provided by a volatile external environment.

4.2 Microservice architectures

A microservices architecture was defined by Fowler *et al.* as an approach to building modular software systems with well-defined domain boundaries [53], which is more suited to enabling evolution of the underlying design. Microservices draw from principles of modular software design [2], and expand this idea from focusing on the modularity of functions, classes, and other code-level components, to a higher-level system component level. They are a response to bulky monolithic systems, where, for example, all back-end logic of an application is centralised in one place, and as such, any change to that back-end requires a complete redeployment of all instances of the monolith [54]. The decoupled nature of microservices encourages a distributed deployment of these systems, serving one of the motivations of the architecture: resilience to failure [53].

These motivations were confirmed by a survey by Viggiato *et al.* [55], where benefits such as independent deployment, ease of scalability, maintainability and technology flexibility were identified by respondents as having some or great importance. It may have been better, however, to ask more open-ended questions around the benefits of microservices instead of the very guided questions reported. Taibi *et al.* also added a motivation of having no other choice with regards to handling system complexity [56], though their survey sample size was small.

Gravanis *et al.* presented a counterargument for the adoption of microservices [57], stating that a rapid adoption of this newfound architecture showcases its own problems (including the complexity of distributed communication, as mentioned by Viggiato *et al.*), and that the monolith can still stand as a valid choice of architecture for businesses. While Fowler *et al.* see decentralised data management as advantageous, Gravanis *et al.* identify this as a non-trivial flaw of the architecture, including issues such as inconsistency of data across a system. However, the primary point made is that choosing a particular software architecture should be informed by the alignment of the intended nature of a system and the benefits provided by said architecture, rather than a dogmatic selection based on current trends. The need for a performant system and the scalability provided by microservices is one such alignment. Scalability was re-oriented as a core feature of microservices by Dragoni *et al.*, with other desired system attributes such as performance, availability and resilience made possible through scalability [58].

An example of premature architecture selection is seen in [59], where a third-party Kubernetes control plane component was initially designed as a set of microservices, but was eventually converted into a traditional monolith after failing to reap the often promoted benefits of microservices. This case for moving to a monolith, however, seems to be rooted in an initial misalignment of the concerns of the developers of the system, versus those who would actually install and deploy said system, which was briefly addressed by Mendonça *et al.* Jamshidi *et al.* instead see the distributed nature of microservices as being a point of caution, requiring attention paid to how well a system under this architecture performs self-healing, which is a non-trivial endeavour, as failure within a distributed system itself becomes potentially distributed [54].

Self-adaption has, over time, been applied in increasing amounts to microservice architectures, as shown by Filho *et al.* [60], where various papers applying different aspects of self-adaptation were analysed. Although this transition has been quite plain, Filho *et al.* seem to ignore the obvious mainstream application of self-adaptation to microservices, Kubernetes, which Jamshidi *et al.* identified as being part of the fourth wave of microservices usage [54].

In an exploration of how microservices perform compared to monoliths, Al-Debagy *et al.* collected experimental results for a small example application with two variants for the two architectures [61]. They did not find any substantial performance differences between the two architectures, though little description of the test scenario was provided, and the entire experiment was run on a single system. These results are contradicted by Aydemir *et al.*, where in [62], a sophisticated banking application was re-designed as a microservices-based system, with the central goal of attaining a more performant system in mind. A monolithic application which originally handled bank accounts, transfers, loans and customer management was restructured to have a microservice for each of those four banking elements. The final system was found to handle substantially higher throughput with less resource utilisation.

When applying principles of modularity to the design of a reliable distributed system, with a need for various independent components to be well-integrated, microservices can be seen as a natural starting approach. However, on their own, reliability cannot be entirely attained without a well-designed approach for handling failure.

4.3 Frameworks for developing self-adaptive distributed systems

While there is a large range of esoteric frameworks and systems available for deploying microservices architectures and ensuring the various self-oriented properties of self-adaptive systems, the two most prominent of these in the literature are Docker and Kubernetes.

4.3.1 Docker

Docker is a container runtime and management engine, making easy the process of creating and running custom containers, as well as using container images published by other individuals or organisations to an open repository of community-made images [63]. It is available for use on all major operating systems for software development, including Linux, macOS and Windows.

Docker was used in a performance comparison between a monolithic architecture and microservices architecture enacted by Al-Debagy *et al.* [61]. Its utilisation in this small-scale experiment can be explained by its ease of use and general pervasive availability on different systems. Saquicela *et al.* use Docker in a more sophisticated fashion, using it to build and run various containerised (third-party and custom-made) microservices, such as a load balancer, service discovery provider, and a Redis database.

With containers being one of the primary means of deploying individual microservices [65], Docker is naturally used by default, as it played a major part in the evolution of container runtimes, building on the foundational container technology of LXC (Linux Containers), and contributing to standardisation of the container image format, OCI (Open Container Initiative) [23].

4.3.2 Kubernetes

Kubernetes is a container orchestration and management system [66] designed to coordinate container-based workloads across multiple machines (a cluster) with ease. A control plane oversees the rest of the cluster, monitoring for failure or poor resource utilisation, restarting failed pods (each pod is a group of tightly-coupled containers) or instantiating more or less of a given pod based on system load. Kubernetes is one of the more concrete mainstream implementations of a self-adaptive system, as outlined by Weyns [52].

As part of their testing of a microservices architecture for a banking system, Aydemir *et al.* deployed their system using Kubernetes, allowing for the system to adapt to variable levels of throughput generated by their load testing tool. [62]. Figueira *et al.* specifically discuss developing self-adaptive

microservices, and describe a system architecture built on top of Kubernetes for its scalability and availability attributes. Randal further presented the picture of Kubernetes as being the mainstream choice for container orchestration, in part for its ability to reduce the complexity involved in managing a distributed software system [23].

While it has not been defined explicitly as the culmination of self-adaptive computing and distributed computing combining, Kubernetes is targeted towards providing engineers with the ability to construct systems capable of autonomously maintaining attributes such as performance and reliability, especially in the context of microservices.

5. Software system performance analysis

When assessing software systems and their respective configurations, there is a range of approaches that can be used. The literature reviewed can be categorised into empirical performance testing and performance modelling. Empirical performance testing involves the experimental collection of performance-related data using active and passive system probing techniques. Performance modelling addresses theoretical results obtained via representations of the underlying system.

5.1 Empirical performance testing

Cheng emphasised the use of appropriate KPIs (Key Performance Indicators) [68] for thoroughly and properly assessing the performance of enterprise software systems. These KPIs can be assessed with variations of end-to-end testing, simulating the behaviour of both a single user (performing some sequence of actions) and multiple users for wider load-testing experiments. With these KPIs, the impact of different optimisations (such as configuration tuning, adding additional hardware, architectural changes, etc.) can be measured by comparing KPI changes before and after experiments. However, Cheng suggested using performance tests for determining explicit hardware allocation guidelines, which may no longer be appropriate when striving for autonomous system management. These guidelines would also fail to consider dynamic system load as a result of variable user activity.

Chuang *et al.* use the Selenium test automation framework to perform active testing and monitoring of a data analysis system built on Kubernetes [69]. They compared the response times of different user requests, and the CPU throttling effects on different system components, before and after configuration tuning. After tuning, they were able to observe a decrease in CPU throttling. It could be considered problematic for accurate performance measurement when the testing tool (Selenium) is run from inside the clustered system being tested, rather than from an external host with separate resources.

The usage of load-testing was expanded upon in [62], where a legacy monolithic banking application was compared against a new microservices architecture deployed on Kubernetes. Apache JMeter was used to create scenarios of between 500 and 5000 virtual users interacting with the banking system. Each load-test was conducted over the course of 5 minutes, with an initial ramp-up of 1 minute before reaching a maximum request-rate, and all were run from an external system. Two system-level metrics were collected during tests: throughput and response time. The impact of different Kubernetes configurations, however, was not considered.
This active testing approach is included in an expanded performance monitoring description given by Nambiar *et al.* [70]. They, however, preferred the usage of passive monitoring to observe performance impacts, as active monitoring or testing introduces additional load to the system subject to analysis. Their passive monitoring recorded properties of packets sent as part of client requests and server responses. It may be difficult to confirm the purported utility of the specific tool used by Nambiar *et al.*, as it is licensed software.

5.2 Performance modelling

An alternative to live testing, monitoring and analysis of a software system is to model or simulate it. This was shown in [71], where a Poisson modelling approach is used to analyze the performance of a serverless AWS (Amazon Web Services) Lambda application. Such an approach can avoid the complication of preparing a suitable testing environment that replicates real-world conditions. To validate the predictions made by the model, additional empirical analysis was conducted. They found that the model was well-aligned with actual data. From this, the utility of modelling a system interaction scenario prior to actual testing is supported.

Simpler pre-experiment modelling was performed in [72] (based on queueing theory), followed by similar subsequent experimental validation. A simple application was designed with both monolithic and microservices architecture variants. The JMeter load-testing tool was used to observe the impact of an increasing number of users on the average response time for each architecture configuration. While the data obtained helped to prove how testing can help make system structure decisions, the application was designed specifically for the study, and as such was trivially implemented, and may lack the complexity to realistically assess the validity of the data.

Woodside presented a case [73] for the integration and dual-use of both system performance models and empirically obtained data to inform decisions. While models will not capture actual data perfectly, they can be fit to previously obtained performance data and utilised for predictions of untested scenarios. However, this does require an established understanding of the underlying model used, as stated by Trivedi *et al.* [74]. Modern software systems are of greater complexity (and increasingly distributed), and thus require more sophisticated models to accurately represent system behaviour. This, in turn, makes the modelling process harder to understand, and as such, it is underutilised within industry. Trivedi *et al.* put that a deep understanding of modelling paradigms, such as queueing theory, Markov chains, and Petri nets, should not be required in order to make productive use of them.

Following review, it can be gathered that the utilisation of both empirical performance testing and performance modelling may be important. Empirical testing serves well to collect actual representations of how a system may perform under various conditions, while modelling techniques can provide predictive power and initial expectations of behaviour, which may be especially useful with complex systems.

6. Summary

A completed review of literature has demonstrated the development and current state for both the engineering of software system architectures, and process simulation software. What has been found is that there is a strong lack of crossover between the modern iterations of both fields. Restated are the original research questions, corresponding answers, and subsequent decisions made with respect to the project at hand:

What architectures are available for deploying and running systems on cloud environments?

A distributed, self-adaptive software system built on Kubernetes, running modular, containerised software components is most suited to deploying systems on cloud environments. The benefits of such an architecture were found to be performance, efficient resource utilisation, and system resilience. The flexible and lightweight nature of containers can serve the project's needs in terms of performance very well, and serve as the basis for all software components involved when deployed. Such flexibility can better enable the analysis of complex systems, especially when the analysis is multi-faceted. In a distributed fashion, the usage of containers will allow a deployed analysis system to efficiently take advantage of computing power provided by an array of heterogeneous hardware.

As such, the project will move towards the described architecture.

What challenges and limitations are being faced by process simulation software?

Interactive process simulation software primarily use solving methods that are challenging to parallelise, and most software is deployed in a standalone, local fashion that cannot take advantage of heterogeneous hardware resources. More performant and reliable mathematical modelling techniques are largely unavailable via such interactive software, preventing their extensive use in industry.

As such, the project will assess the specific performance improvements obtained by deploying an integrated process engineering software platform to the aforementioned architecture.

What tools or techniques have been used for assessing the performance of different software architectures?

Both empirical performance testing and performance modelling methods have been used to assess software systems, including active load-testing and passive monitoring as empirical methods, and Markov chains and Petri nets as modelling methods.

As such, the project will include the usage of well-defined performance modelling methods in addition to empirical analysis techniques to assess the performance of the developed software architecture.

References

[1] K.-U. Klatt and W. Marquardt, "Perspectives for process systems engineering—personal views from academia and industry," *Computers & Chemical Engineering*, Selected Papers from the 17th European Symposium on Computer Aided Process Engineering held in Bucharest, Romania, May 2007, vol. 33, no. 3, pp. 536–550, Mar. 20, 2009.

[2] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1, 1972.

[3] D. Parnas, P. Clements, and D. Weiss, "The modular structure of complex systems," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 3, pp. 259–266, Mar. 1985.

[4] J. D. McGregor, "Software architecture.," *The Journal of Object Technology*, vol. 3, no. 5, p. 65, 2004.

[5] E. de Almeida, A. Alvaro, D. Lucredio, V. Garcia, and S. de Lemos Meira, "RiSE project: Towards a robust framework for software reuse," in *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, 2004. IRI 2004.*, Nov. 2004, pp. 48–53.

[6] J. Sifakis, "A framework for component-based construction," in *Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)*, Sep. 2005, pp. 293–299.

[7] N. Goncalves, D. Faustino, A. R. Silva, and M. Portela, "Monolith modularization towards microservices: Refactoring and performance trade-offs," in *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, Stuttgart, Germany: IEEE, Mar. 2021, pp. 1–8.

[8] Haynes, Lau, Siewiorek, and Mizell, "A survey of highly parallel computing," *Computer*, vol. 15, no. 1, pp. 9–24, Jan. 1982.

[9] K. Asanovic, R. Bodik, B. C. Catanzaro, *et al.*, "The landscape of parallel computing research: A view from berkeley," Dec. 18, 2006.

[10] M. Bridges, N. Vachharajani, Y. Zhang, T. Jablin, and D. August, "Revisiting the sequential programming model for multi-core," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, Dec. 2007, pp. 69–84.

[11] K. Asanovic, R. Bodik, J. Demmel, *et al.*, "A view of the parallel computing landscape," *Communications of the ACM*, vol. 52, no. 10, pp. 56–67, Oct. 1, 2009.

[12] B. K. Harrison, "Computational inefficiencies in sequential modular flowsheeting," *Computers & Chemical Engineering*, An International Journal of Computer Applications in Chemical Engineering, vol. 16, no. 7, pp. 637–639, Jul. 1, 1992.

[13] G.-L. Chen, G.-Z. Sun, Y.-Q. Zhang, and Z.-Y. Mo, "Study on parallel computing," *Journal of Computer Science and Technology*, vol. 21, no. 5, pp. 665–673, 2006.

[14] Z. Xu, Y. He, W. Lin, and L. Zha, "Four styles of parallel and net programming," *Frontiers of Computer Science in China*, vol. 3, no. 3, pp. 290–301, Sep. 2009.

[15] F. Seveso, R. Marichal, E. Dufrechou, and P. Ezzatti, "Refactoring an electric-market simulation software for massively parallel computations," in *High Performance Computing*, Springer, Cham, 2022, pp. 190–204.

[16] A. Vivas and H. Castro, "Quantitative characterization of scientific computing clusters," in *High Performance Computing*, Springer, Cham, 2022, pp. 47–62.

[17] R. P. Goldberg, "Survey of virtual machine research," *Computer*, vol. 7, no. 6, pp. 34–45, Jun. 1974.

[18] A. Whitaker, M. Shaw, and S. D. Gribble, "Denali: Lightweight virtual machines for distributed and networked applications," 2001.

[19] Y. Li, W. Li, and C. Jiang, "A survey of virtual machine system: Current technology and future trends," in *2010 Third International Symposium on Electronic Commerce and Security*, Jul. 2010, pp. 332–336.

[20] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, "Containers and virtual machines at scale: A comparative study," in *Proceedings of the 17th International Middleware Conference*, ser. Middleware '16, New York, NY, USA: Association for Computing Machinery, Nov. 28, 2016, pp. 1–13.

[21] I. Pietri and R. Sakellariou, "Mapping virtual machines onto physical machines in cloud computing: A survey," *ACM Computing Surveys*, vol. 49, no. 3, pp. 1–30, Sep. 30, 2017.

[22] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.

[23] A. Randal, "The ideal versus the real: Revisiting the history of virtual machines and containers," *ACM Computing Surveys*, vol. 53, no. 1, 5:1–5:31, Feb. 6, 2020.

[24] A. Rosenthal, P. Mork, M. H. Li, J. Stanford, D. Koester, and P. Reynolds, "Cloud computing: A new business paradigm for biomedical information sharing," *Journal of Biomedical Informatics*, vol. 43, no. 2, pp. 342–353, Apr. 1, 2010.

[25] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: A state-of-the-art review," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, Jul. 2019.

[26] I. Ahmad, M. G. AlFailakawi, A. AlMutawa, and L. Alsalman, "Container scheduling techniques: A survey and assessment," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 7, pp. 3934–3947, Jul. 1, 2022.

[27] P.-J. Maenhaut, B. Volckaert, V. Ongenae, and F. De Turck, "Resource management in a containerized cloud: Status and challenges," *Journal of Network and Systems Management*, vol. 28, no. 2, pp. 197–246, Apr. 1, 2020.

[28] R. Vaño, I. Lacalle, P. Sowiński, R. S-Julián, and C. E. Palau, "Cloud-native workload orchestration at the edge: A deployment review and future directions," *Sensors*, vol. 23, no. 4, p. 2215, 2023.

[29] V. A. Merchan, E. Esche, S. Fillinger, G. Tolksdorf, and G. Wozny, "Computer-aided process and plant development. a review of common software tools and methods and comparison against an integrated collaborative approach," *Chemie Ingenieur Technik*, vol. 88, no. 1, pp. 50–69, 2016.

[30] L. T. Biegler, "Recent advances in chemical process optimization," *Chemie Ingenieur Technik*, vol. 86, no. 7, pp. 943–952, 2014.

[31] S. Begum, M. G. Rasul, D. Akbar, and N. Ramzan, "Performance analysis of an integrated fixed bed gasifier model for different biomass feedstocks," *Energies*, vol. 6, no. 12, pp. 6508–6524, Dec. 2013.

[32] Q. Smejkal and M. Šoóš, "Comparison of computer simulation of reactive distillation using aspen plus and hysys software," *Chemical Engineering and Processing: Process Intensification*, vol. 41, no. 5, pp. 413–418, May 1, 2002.

[33] N. M. A. Al-Lagtah, S. Al-Habsi, and S. A. Onaizi, "Optimization and performance improvement of lekhwair natural gas sweetening plant using aspen HYSYS," *Journal of Natural Gas Science and Engineering*, vol. 26, pp. 367–381, Sep. 1, 2015.

[34] R. Gani, "Chemical product design: Challenges and opportunities," *Computers & Chemical Engineering*, vol. 28, no. 12, pp. 2441–2457, Nov. 15, 2004.

[35] T. Ralphs, Y. Shinano, T. Berthold, and T. Koch, "Parallel solvers for mixed integer linear optimization," in *Handbook of Parallel Constraint Reasoning*, 1st ed., Springer Cham, Apr. 6, 2018, pp. 283–336.

[36] V. Kulikov, H. Briesen, R. Grosch, A. Yang, L. von Wedel, and W. Marquardt, "Modular dynamic simulation for integrated particulate processes by means of tool integration," *Chemical Engineering Science*, vol. 60, no. 7, pp. 2069–2083, Apr. 1, 2005.

[37] N. Abdel-Jabbar, B. Carnahan, and C. Kravaris, "A multirate parallel-modular algorithm for dynamic process simulation using distributed memory multicomputers," *Computers & Chemical Engineering*, vol. 23, no. 6, pp. 733–761, Jun. 1, 1999.

[38] E. N. Pistikopoulos, A. Barbosa-Povoa, J. H. Lee, *et al.*, "Process systems engineering – *The generation next?*" *Computers & Chemical Engineering*, vol. 147, p. 107 252, Apr. 1, 2021.

[39] M. Martín and T. A. Adams II, "Challenges and future directions for process and product synthesis and design," *Computers & Chemical Engineering*, vol. 128, pp. 421–436, Sep. 2, 2019.

[40] J. De Tommaso, F. Rossi, N. Moradi, C. Pirola, G. S. Patience, and F. Galli, "Experimental methods in chemical engineering: Process simulation," *The Canadian Journal of Chemical Engineering*, vol. 98, no. 11, pp. 2301–2320, 2020.

[41] S. Cremaschi, "A perspective on process synthesis: Challenges and prospects," *Computers & Chemical Engineering*, Special Issue: Selected papers from the 8th International Symposium on the Foundations of Computer-Aided Process Design (FOCAPD 2014), July 13-17, 2014, Cle Elum, Washington, USA, vol. 81, pp. 130–137, Oct. 4, 2015.

[42] M. A. Duran and I. E. Grossmann, "A mixed-integer nonlinear programming algorithm for process systems synthesis," *AIChE Journal*, vol. 32, no. 4, pp. 592–606, Apr. 1986.

[43] P. Nayak, P. Dalve, R. A. Sai, *et al.*, "Chemical process simulation using OpenModelica," *Industrial & Engineering Chemistry Research*, vol. 58, no. 26, pp. 11 164–11 174, Jul. 3, 2019.

[44] G. Stephanopoulos and G. V. Reklaitis, "Process systems engineering: From solvay to modern bio- and nanotechnology.: A history of development, successes and prospects for the future," *Chemical Engineering Science*, Multiscale Simulation, vol. 66, no. 19, pp. 4272–4306, Oct. 1, 2011.

[45] A. W. Dowling and L. T. Biegler, "A framework for efficient large scale equation-oriented flowsheet optimization," *Computers & Chemical Engineering*, A Tribute to Ignacio E. Grossmann, vol. 72, pp. 3–20, Jan. 2, 2015.

[46] Y. Ma, Z. Shao, X. Chen, and L. T. Biegler, "A parallel function evaluation approach for solution to large-scale equation-oriented models," *Computers & Chemical Engineering*, vol. 93, pp. 309–322, Oct. 4, 2016.

[47] D. D. Nikolic, "DAE tools: Equation-based object-oriented modelling, simulation and optimisation software," *PeerJ Computer Science*, vol. 2, e54–e54, Apr. 6, 2016.

[48] D. Bongartz and A. Mitsos, "Deterministic global flowsheet optimization: Between equation-oriented and sequential-modular methods," *AIChE Journal*, vol. 65, no. 3, pp. 1022–1034, 2019.

[49] R. C. Pattison, A. M. Gupta, and M. Baldea, "Equation-oriented optimization of process flowsheets with dividing-wall columns," *AIChE Journal*, vol. 62, no. 3, pp. 704–716, 2016.

[50] Y. Kang, Y. Luo, and X. Yuan, "Recent progress on equation-oriented optimization of complex chemical processes," *Chinese Journal of Chemical Engineering*, vol. 41, pp. 162–169, Jan. 1, 2022.

[51] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[52] D. Weyns, *Engineering Self-Adaptive Software Systems – An Organized Tour*. Sep. 1, 2018.

[53] M. Fowler and J. Lewis. "Microservices - a definition of this new architectural term," martinfowler.com. (Mar. 25, 2014), [Online]. Available: `https://martinfowler.com/articles/microservices.html` (visited on 04/09/2024).

[54] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, May 2018.

[55] M. Viggiato, R. Terra, H. Rocha, M. T. Valente, and E. Figueiredo, *Microservices in practice: A survey study*, Aug. 14, 2018.

[56] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, Sep. 2017.

[57] D. Gravanis, G. Kakarontzas, and V. Gerogiannis, "You don't need a microservices architecture (yet): Monoliths may do the trick," in *Proceedings of the 2021 European Symposium on Software Engineering*, ser. ESSE '21, New York, NY, USA: Association for Computing Machinery, Mar. 26, 2022, pp. 39–44.

[58] N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina, "Microservices: How to make your application scale," in *Perspectives of System Informatics*, A. K. Petrenko and A. Voronkov, Eds., vol. 10742, Cham: Springer International Publishing, 2018, pp. 95–104.

[59] N. C. Mendonça, C. Box, C. Manolache, and L. Ryan, "The monolith strikes back: Why istio migrated from microservices to a monolithic architecture," *IEEE Software*, vol. 38, no. 5, pp. 17–22, Sep. 2021.

[60] M. Filho, E. Pimentel, W. Pereira, P. H. M. Maia, and M. I. Cortés, "Self-adaptive microservice-based systems - landscape and research opportunities," in *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, May 2021, pp. 167–178.

[61] O. Al-Debagy and P. Martinek, "A comparative review of microservices and monolithic architectures," in *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, Nov. 2018, pp. 000 149–000 154.

[62] F. Aydemir and F. Başçiftçi, "Building a performance efficient core banking system based on the microservices architecture," *Journal of Grid Computing*, vol. 20, no. 4, p. 37, Nov. 2, 2022.

[63] "Why docker — docker." (Nov. 11, 2021), [Online]. Available: `https://www.docker.com/why-docker/` (visited on 04/29/2024).

[64] V. Saquicela, G. Campoverde, J. Avila, and M. E. Fajardo, "Building microservices for scalability and availability: Step by step, from beginning to end," in *New Perspectives in Software Engineering*, J. Mejia, M. Muñoz, Á. Rocha, and Y. Quiñonez, Eds., vol. 1297, Cham: Springer International Publishing, 2021, pp. 169–184.

[65] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, "Design, monitoring, and testing of microservices systems: The practitioners' perspective," *Journal of Systems and Software*, vol. 182, p. 111 061, Dec. 1, 2021.

[66] "Kubernetes." (), [Online]. Available: `https://kubernetes.io/` (visited on 04/29/2024).

[67] J. Figueira and C. Coutinho, "Developing self-adaptive microservices," *Procedia Computer Science*, 5th International Conference on Industry 4.0 and Smart Manufacturing (ISM 2023), vol. 232, pp. 264–273, Jan. 1, 2024.

[68] X. Cheng, "Performance, benchmarking and sizing in developing highly scalable enterprise software," in *Performance Evaluation: Metrics, Models and Benchmarks*, S. Kounev, I. Gorton, and K. Sachs, Eds., Berlin, Heidelberg: Springer, 2008, pp. 174–190.

[69] C.-C. Chuang and Y.-C. Tsai, "Performance evaluation and improvement of a cloud-native data analysis system application," in *2021 International Conference on Electronic Communications, Internet of Things and Big Data (ICEIB)*, Dec. 2021, pp. 60–62.

[70] M. Nambiar and H. K. Kalita, "Performance monitoring and analysis of a large online transaction processing system," in *Performance Evaluation: Metrics, Models and Benchmarks*, S. Kounev, I. Gorton, and K. Sachs, Eds., Berlin, Heidelberg: Springer, 2008, pp. 303–313.

[71] N. Mahmoudi and H. Khazaei, "Performance modeling of serverless computing platforms," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2834–2847, Oct. 2022.

[72] M. Jagiełło, M. Rusek, and W. Karwowski, "Performance and resilience to failures of an cloud-based application: Monolithic and microservices-based architectures compared," in *Computer Information Systems and Industrial Management*, K. Saeed, R. Chaki, and V. Janev, Eds., Cham: Springer International Publishing, 2019, pp. 445–456.

[73] M. Woodside, "The relationship of performance models to data," in *Performance Evaluation: Metrics, Models and Benchmarks*, S. Kounev, I. Gorton, and K. Sachs, Eds., Berlin, Heidelberg: Springer, 2008, pp. 9–28.

[74] K. S. Trivedi, B. R. Haverkort, A. Rindos, and V. Mainkar, "Techniques and tools for reliability and performance evaluation: Problems and perspectives," in *Computer Performance Evaluation Modelling Techniques and Tools*, G. Haring and G. Kotsis, Eds., Berlin, Heidelberg: Springer, 1994, pp. 1–24.

Appendices

The project proposal can be found on the next page.

ENGEN582-24X
Honours Research and Development Project

Integration of cloud computing paradigms for performant analysis of simulated process engineering applications.

Caleb Archer

Supervised by Tim Walmsley, Mark Apperley

Proposal Summary

With new modes of software system organisation and management available to organisations, which help to deliver efficient and scalable outcomes, the process engineering computing space is seeing itself left behind, suffering from both a stunted speed of analysis and a lack of centralised data integration. This project will move the Ahuora Digital Platform onto a distributed Kubernetes cluster hosted on Raspberry Pi nodes, assessing the performance consequences of this move, as well as determine the ideal set of policies with respect to performance outcomes, building on the foundations of existing research. Given the existing access to the necessary compute power, networking components, and free software, the progress of cluster research and development is not dependent on acquisition of further resources. Completion of this project will provide a basis for the execution of more efficient, sustainable, and speedy process simulation and analysis, while taking another step towards industry decarbonisation.

1. Background

Computing and access to computing has changed substantially in the past two decades. Cloud computing is one of the main drivers of this change, and continues to grow rapidly [1]. There has been a shift from software that is run entirely on local devices, paid for via costly perpetual licenses, to software-as-a-service styles of delivery, often made available at a lower, ongoing cost. The complexity involved in managing this new model of software systems, dependent on multiple components (distributed software) to control and deliver the product, has seen tools developed in order to manage this complexity [2]. One such tool is Kubernetes, a container orchestration and management platform built to improve system efficiency and better enable complex software systems to scale according to fluctuating needs. These two properties, efficiency and scalability, provide the core motivation for the Ahuora Digital Platform to deploy on top of Kubernetes.

A 2019 CACHE (Computer Aids for Chemical Engineering) Corporation survey [3] indicated that the proportion of the respondents from the chemical and process engineering industry using numerical analysis software had risen from 26% in 2003 to 38%. Likewise, the proportion of respondents reporting usage of process simulation software had risen from approximately 34% to almost 50%. The two most common process simulation tools were, by a large margin, Aspen Plus and Aspen Hysys.

With the Aspen process simulation tools representing the majority of process simulation software usage amongst chemical and process engineers, it is clear that the industry has a dependency on standalone, closed-source, monolithic software, and it lags in the adoption of modern cloud computing paradigms, tools, and techniques. F. Tapia et al determined that a microservices architecture provides benefits with respect to hardware resource efficiency, costs and productivity over a traditional monolithic software architecture [4]. The chemical and process engineering industry is missing out on these benefits, especially given the compute and time-intensive nature of process simulation and analysis tasks. Monolithic simulation and analysis systems are more sensitive to the fidelity-speed tradeoff, where simulations of higher (desirable) fidelity take longer to complete compared to a result of lower fidelity [5].

The software usage habits of the chemical and process engineering industry also face centralisation issues. By the nature of monolithic, locally deployed software, it is difficult to maintain a centralised and shared view of an entire process plant or site, as information may be available only on one licensed machine, or spread amongst many. With the usage of a distributed, cloud-native architecture, it would be easier to centralise data access and improve the efficiency of information-sharing amongst a process-centred organisation.

The end vision of the Ahuora Digital Platform sees a wide range of process tools being integrated in a centralised fashion, including hybrid pinch, P-graph, deterministic and stochastic optimisation, physics and machine learning-based simulation, data streaming, MPC control and scheduling, and more. This desired integration will be easier to realise by employing Kubernetes as a system management foundation.

2. Overall Aim of the Project

The goal of the project is to construct and test, on top of a physical cluster of Raspberry Pi nodes, a performant, self-adaptive and self-healing distributed system architecture for the Ahuora Digital Platform. This shall involve utilisation of the Kubernetes container orchestration and cluster management platform to deploy, manage, and scale the various components of the platform. To achieve this, load-balancing and scaling policies need to be controlled in such a way that the system meets such performance requirements. Research involving comparative performance analysis of such policies will need to be carried out to inform policy choices.

The finished system will need to efficiently enable process engineering simulations to be carried out, while appropriately managing total consumption of available compute power and memory. It should be possible to automatically deploy and configure the entire Kubernetes cluster. Along with this, the integrity of data stored across a distributed PostgreSQL database must be preserved. This project will ultimately culminate in access to flexible, performant and fault-tolerant analysis of complex processing systems over multiple operating contexts or system parameter configurations.

3. Research and Development Plan

Progress made towards achieving project outcomes will be broken into an interconnected series of stages, some more focused on research, and others development.

The first major research component will involve conducting a literature and technology review in order to assess the previous work and findings of other researchers in the fields of software system performance analysis, distributed system orchestration and configuration, and the cross-section of these.

The existing system components of the Ahuora Digital Platform be migrated towards fully containerised implementations, making them ready for initial deployment to the cluster. These containerised processes, along with cluster deployments, will have baseline policies put in place, which will then be refined further in the research and development process.

Throughout the development process, automated infrastructure management tools, such as Ansible, will be employed to quasi-deterministically create and configure the systems on which the cluster will run, as well as the cluster configuration itself. Such tools are required to reduce the impact of error-prone, manual configuration management, which may also be hard to replicate. This will simplify the process of duplicating the ideal state of the cluster across different physical cluster environments.

Experiment-based comparative analysis of possible load-balancing, scaling, resource allocation and topological arrangement policies shall be conducted to determine the optimal configuration of the Kubernetes cluster. This will be in the form of performance testing, where traffic and request load simulations will be executed, and key metrics shall be measured using a range of monitoring tools, and compared to a baseline measurement across different configuration strategies and policy choices. Experiments will be constructed with well-defined test-cases, which will identify the variable to assess

and modify, as well as variables to keep constant. Similarly structured experiments will also be used to infer any system bottlenecks, allowing decisions to be made on the ideal granularity of containerised processes, especially with respect to compute-heavy analysis components. Any analysis will take into consideration previous work identified in the aforementioned literature review.

Research into the advantages and disadvantages provided by centralised versus distributed software systems shall be conducted. This shall involve direct comparisons of performance achieved between distributed and local versions of the same software system.

There are elements of the research process that create a level of uncertainty, which may delay or otherwise hinder project progress. For example, the system bottleneck analysis assessing containerisation granularity could result in slow progress, depending on what stage of development each process analysis component is in. If it is determined that two components could be containerised together without any performance impacts, there may be a large delay in being able to action this decision, since this would depend on other engineers within Ahuora to make the necessary changes, which may or may not be substantial. Another source of uncertainty is the accuracy of the performance measurement tools used, and the impact they may have on results. Along with this, even with confidence in measurement accuracy, results may be skewed regardless due to the performance overheads from conducting such measurements.

4. Resources

As a software-focused project, there is no substantial dependency on many different physical resources in order to conduct the research and development process of the project.

There is already access to eight Raspberry Pi 5 devices, on which the Kubernetes cluster will be deployed and configured. Along with this, there is a 4G router for providing internet access to the cluster (independent of the primary university network), an eight-port network switch, ethernet cables, and a rack for storing the cluster nodes. Necessary IDEs (Integrated Development Environments) and code editors such as IntelliJ PyCharm and Visual Studio Code are already installed on a dedicated office workstation. Any unforeseen need for particular software is likely to be met by additional open-source software.

One of the more probable risks that could impact project progress is failure of one or more of the Raspberry Pi nodes that shall make up the cluster. Although there could be performance impacts due to a lower node count, cluster operation and development could easily continue, as one of the primary features provided by Kubernetes is resilience to node failure.

A greater problem that could present itself would be failure of a control node (a node partially or fully responsible for managing the rest of the cluster), especially if there is only one control node. Failure of a lone control node could paralyze the entire cluster. Mitigations include: usage of the configuration automation tool Ansible (whereby the cluster set-up process is defined in code), designating at least three control nodes to the cluster for high availability, and regular back-ups of critical cluster data.

In general, replacing a failed Raspberry Pi would not be at great cost, as a single Raspberry Pi 5 costs approximately $160 NZD, including a new active cooler.

5. Sustainability and Vision Mātauranga

Through the usage of a performance and resource-optimised Kubernetes cluster designed for enabling efficient and effective process simulation and analysis within process engineering contexts, sustainable

computational resource usage is greatly enhanced. Energy already provided to computers on which process software is run can be driven to high efficiency and utilisation, and consequently, process engineers can gain access to the results they need sooner, and therefore make decisions to optimise a particular process faster.

Though this project does not directly invoke integration of Vision Mātauranga principles, it does contribute to the wider Ahuora Digital Platform aspiration of promoting kaitiatikanga (guardianship) and responsible stewardship of the environment, through providing the process engineering industry the digital tools they need to decarbonise and make more efficient their operations.

References

[1] Mordor Intelligence. Cloud computing market - size, growth, report & analysis, n.d. https://www.mordorintelligence.com/industry-reports/cloud-computing-market.

[2] Kubernetes. Overview, n.d. https://kubernetes.io/docs/concepts/overview/.

[3] Robert P. Hesketh, Martha Grover, and David L. Silverstein P.E. Cache/asee survey on computing in chemical engineering. In *2020 ASEE Virtual Annual Conference Content Access*, number 10.18260/1-2–34249, Virtual On line, June 2020. ASEE Conferences. https://peer.asee.org/34249.

[4] Freddy Tapia, Miguel Ángel Mora, Walter Fuertes, Hernán Aules, Edwin Flores, and Theofilos Toulkeridis. From monolithic systems to microservices: A comparative study of performance. *Applied Sciences*, 10(17), 2020. https://www.mdpi.com/2076-3417/10/17/5797.

[5] sJie Xu, sSi Zhang, sEdward Huang, sChun-Hung Chen, sLoo Hay Lee, and sNurcin Celik. Efficient multi-fidelity simulation optimization. In *Proceedings of the Winter Simulation Conference 2014*, pages 3940–3951, 2014.

Appendices

A. Gantt Chart