Kalpa Publications in Computing

Volume 17, 2023, Pages 31–47

Proceedings of 2023 Concurrent Processes Architectures and Embedded Systems Hybrid Virtual Conference

Concurrency and Models of Abstraction: Past, Present and Future

Jeremy M. R. Martin
Lloyd's of London
One Lime Street
London EC3M 7HA
UK

Abstract. I will present a personal view of some of the key historical developments in Concurrency Theory and how abstract models have been used to make it easier to develop concurrent systems. I will then provide an assessment of certain concurrency issues facing us today and make predictions as to how these will be solved in the future.

Keywords—Concurrency, Abstraction, CSP, Map-Reduce, Microservices, Data Lake, Quantum Computing, Cyber Terrorism.

I. Introduction

Concurrency in a computer system is the phenomenon of multiple computations and processes happening at the same time [1,2]. It is an inevitable feature of many problem domains that will need to be embraced when designing solutions. However, in its most general and complex form, it presents serious challenges to software and hardware engineers to understand how to manage it and be able to deliver reliable and efficient systems.

Therefore there has been a long history of developing abstract models of concurrency for specific problem domains, which conceal certain details and focus only on the essential concepts needed to be understood by engineers in each particular field. Providing engineers with a suitably abstract model can significantly aid their intuitive understanding of their problem domain and thereby enable them confidently to create extremely complex systems which are nonetheless reliable and effective.

An iconic early example of using abstraction to aid comprehension of a complex concurrent system was the use of the iconic London Underground Map which concealed the geographical complexity of the actual system behind a topologically-equivalent but easy-to-follow abstract map.

In this paper, I shall present an overview of some successful concurrency abstractions which I shall argue have supported major technical advances and efficiencies from which we benefit today.

II. Some key concurrency abstractions from the past

A. The Internet

Arguably the most significant engineering achievement of the past sixty years has been the development of the internet, which now connects billons of people with each other and with billions of devices, enabling rapid exchange of digital information in many different formats. This was built about foundations of telegraphy going back to the early nineteenth century.

How could it be that such a complex system, constantly being modified and added to, relied upon by so many for critical services, can perform so well?

One of the major building blocks is the Internet Protocol Suite [3], which abstracts the internet into four layers. From the bottom up these layers are called Link, Network, Transport and Application. Each one provides services for connecting machines and people. The services of a particular layer are consumed by the services from the layer above, and in turn also consume services from the layer below.

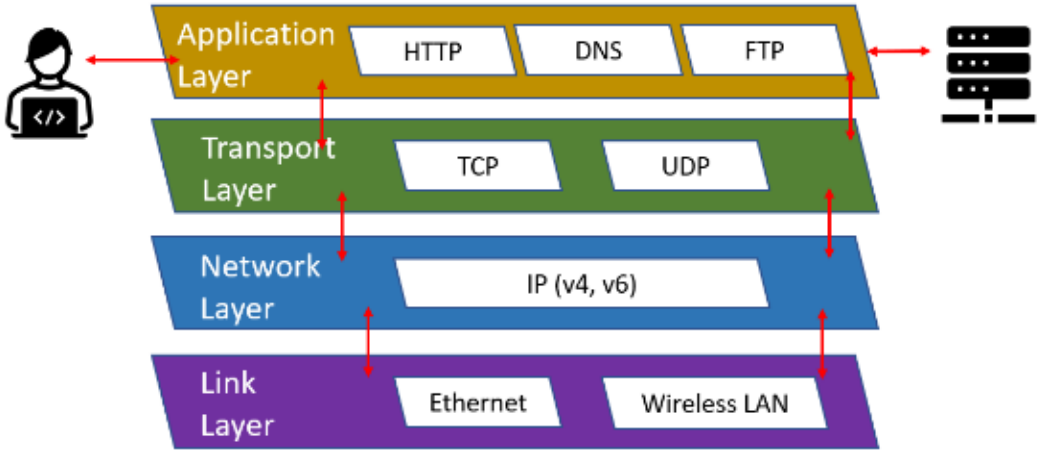

Figure 1. Internet Protocol Suite

The Link Layer provides services which control the physical aspects of network communication. It includes protocols and technologies that govern the transmission of data over a specific physical medium, such as Ethernet, Wi-Fi, or fibre optic.

The Network Layer is responsible for routing data packets between different networks. The Internet Protocol is used for addressing and routing data packets so they can traverse multiple networks to reach their destination. Every device on the internet is given a unique address which is used for sending and receiving messages.

The Transport Layer is responsible for end-to-end communication – it has to ensure that data is *reliably* transmitted between devices on different networks.

The Application Layer deals with end-user applications and services. It includes a wide range of protocols such as the web browsing, email, and file transfer.

The Internet Protocol Suite is highly modular and flexible, allowing different layers to evolve independently and supporting a wide range of applications and technologies. It has been instrumental in the success of the Internet, perhaps the most complex man-made structure built in history. And yet, in essence, it has a very simple abstract structure, which can be described by four layers of protocols, each providing client-server[19] connections to the layer above

B. Microprocessor Architecture

The success of the internet has been underpinned by an exponential increase in the performance and complexity of computer processors. A modern processor may contain as many as 20 billion transistors. And process instructions at a rate of 100 billion per second. This requires incredibly precise engineering and manufacturing to be performed at a microscopic level, where quantum mechanics effects come strongly into view, and yet with the resulting systems reliably performing complex digital operations at breath taking speed.

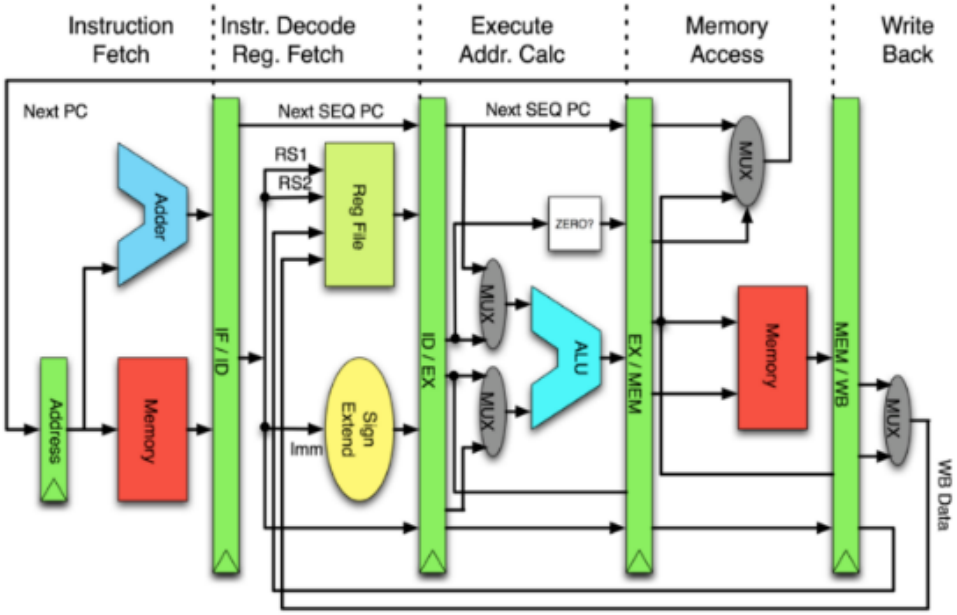

Figure 2. Basic MIPS Architecture

A very important abstract pattern which has supported the rapid evolution of microprocessors is MIPS architecture (Microprocessor without Interlocked Pipeline Stages) [4]. Its simplicity and elegance has made chip design easier in several ways, for example:

- *Reduced Instruction Set Computing (RISC)* - the instruction set architecture contains a limited number of instructions, each of which performs a specific and simple operation, hence reducing the complexity of executing instructions in hardware.
- *Fixed Instruction Length* - instructions are of uniform length (typically 32 bits). This fixed-length encoding simplifies the instruction fetch and decode stages of the pipeline, making it easier to design and implement.
- *Load-Store Architecture* - all operations are performed on registers, and memory operations are limited to load and store instructions. This simplifies the data path and control logic, as there are separate paths for data transfers between registers and memory.

- *Pipelined Architecture* - the architecture is structured into a pipeline with five stages (instruction fetch, instruction decode, execute, memory access, and write-back). This pipelining simplifies the design by breaking down instruction execution into smaller, simple sequential stages. It also allows multiple instructions to be passing through the processor simultaneously to increase the efficiency of the computation and keep each of the five pipeline stages constantly busy.
- *Compiler Friendliness* - the simplicity of the instruction set and architecture makes it easier for compilers to generate highly efficient code

These and other factors made the MIPS pattern a widely used abstract design in the highly competitive and specialised domain of Chip Design, leading to rapid progress in microprocessor technology.

C. Parallel Scientific Computing

Another bastion of complex concurrent systems lies within the world of Parallel Scientific Computing. Scientists and Engineers harness state of the art Supercomputers and Computational Grids to perform simulations of physical systems as accurately and as fast as possible, pushing processors, networks and storage systems to their limits for extended periods.

The PRAM (Parallel Random Access Machine) model[11] provides an abstract model of parallel computation that simplifies the analysis of parallel algorithms by assuming an idealized parallel machine with shared memory and a fixed number of processors.

OpenMP (Open Multi-Processing) is a widely used programming abstraction for the development of parallel programs in shared-memory multiprocessing environments, such as the PRAM. It extends C, C++, and Fortran programming languages with parallel *directives* (expressed as comments within the code) to manage multithreaded execution.

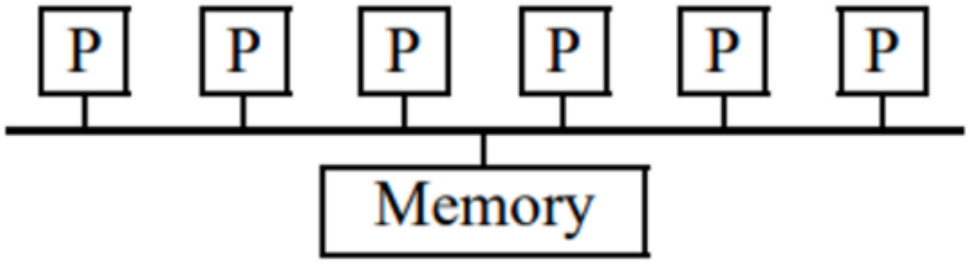

Figure 3. Parallel Random Access Machine

For example, here is a simple Fortran program for matrix addition with an OpenMP parallel loop directive.

```
SUBROUTINE add(left right,out)
REAL(KIND=dp), INTENT(IN) :: left(:,:), right(:,:)
REAL(KIND=dp), INTENT(OUT) :: out(:,:)
```

```
INTEGER :: i, j
!$OMP PARALLEL DO
DO j = 1, UBOUND(out,2)
  DO i = 1, UBOUND(out,1)
    out(i,j) = left(i,j)+right(i,j)
  END DO
END DO
!$OMP END PARALLEL DO
END SUBROUTINE add
```

The OpenMP directive tells the compiler that the commands within the two nested loops may be run concurrently as these are independent calculations. (Unfortunately it is possible to add a directive to a program that will make the answer wrong if you are not careful. It is really important to ensure that there are no dependencies between the commands being executed in parallel!)

The emergence of OpenMP, PRAM, and other similar abstract models for Parallel Scientific Computing, including MPI and BSP, has enabled scientists to create high performance simulation programs that run on the world's most powerful computers without having to understand all the complex details of the underlying machines. This has helped to accelerate research and development in many diverse areas, such as Medicine, Human Genetics, Astronomy, Seismology, and Defence. It is another example of how abstraction can free-up and empower people to achieve extraordinary things.

D. Structured Query Language and Transactions

The world is full of data which need to be processed, updated, protected, transmitted and validated. A very significant development in the field of Data Management has been the definition of the Structured Query Language. This is essentially a textual representation of two branches of Mathematics: Relational Algebra and Relational Calculus, which enables its users to carry out precise and complex mathematic operations on sets of data without having to use any mathematic notation. So SQL in itself is an abstraction away from mathematical notation to code, e.g.

```
SELECT name
FROM players;
WHERE age BETWEEN 18 and 25
AND club = 'Chelsea'
```

The concurrency factor is that databases are often accessed by many people or processes at the same time. For instance consider two people trying to book the last pair of tickets for a theatre show at the same time.

Potentially using a website, each person will log in to see which tickets are available, see the two that are available and then purchase them. Depending on how this is implemented, there could be a problem where the tickets are sold to both people by mistake. And this is where the concept of transactions is useful.

The system can create a transaction for each person which could manage this conflict should the need arise, by use of locks and rollbacks and other concurrency controls.

Using SQL transactions, the consistency of a system can be maintained, even though there may be many users reading it and updating it simultaneously.

Transactions are used to group database operations, expressed in SQL, into a single unit. They are started with a BEGIN TRANSACTION statement and are concluded with a COMMIT or ROLLBACK statement. The ACID properties (Atomicity, Consistency, Isolation, Durability) ensure that transactions are reliable and maintain data consistency. Locks are used to control access to data during transactions. They prevent multiple transactions from simultaneously modifying the same data item, which could lead to conflicts and data inconsistencies. Once a transaction is committed, the changes made by that transaction must be made permanent and durable. If two or more transactions attempt to modify the same data simultaneously, a conflict can occur.

Deadlocks can occur when transactions wait indefinitely for each other to release locks. Relational databases employ deadlock detection and resolution mechanisms to detect and break deadlocks[20], allowing the affected transactions to continue.

Additional complications arise for transactions across distributed collections of databases, for which distributed protocols have been developed for maintaining consistency.

By employing these mechanisms, relational databases ensure that concurrent queries and updates maintain data consistency and prevent corruption due to simultaneous access by multiple users or transactions.

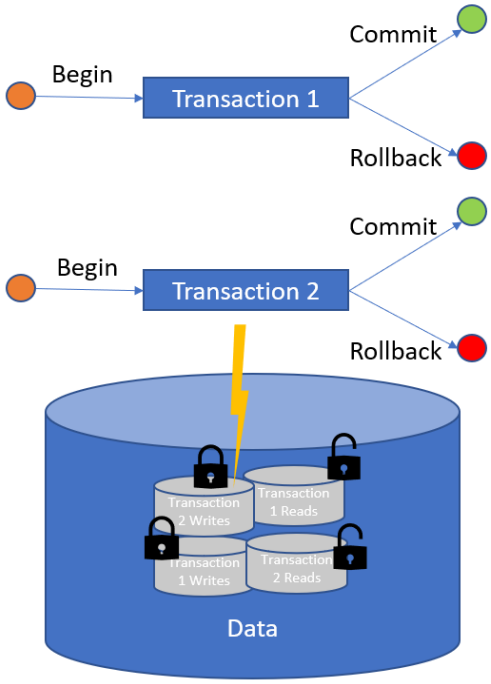

Figure 4. SQL Concurrent Transaction Management

E. Simple Network Management Protocol

We have already discussed how the Internet Protocol Suite has been a huge factor in driving the huge expansion of the Internet in recent times. We shall now focus on one particular protocol which provides

a simple, abstract model to help ensure the smooth operation of the internet - *managing and monitoring network devices*, such as routers, switches, servers, printers.

The Simple Network Management Protocol (SNMP) is a widely used protocol which allows network administrators to collect information and manage network devices efficiently using a simple client-server model [12, 19].

SNMP follows a manager-agent model, where there are two primary components. The SNMP Manager is a software application that runs on a network management system (NMS). It is responsible for sending SNMP requests to devices and receiving responses from those requests or asynchronous alerts. The SNMP Agent is embedded in network devices and provides information about the device's status, configuration, and performance to the manager.

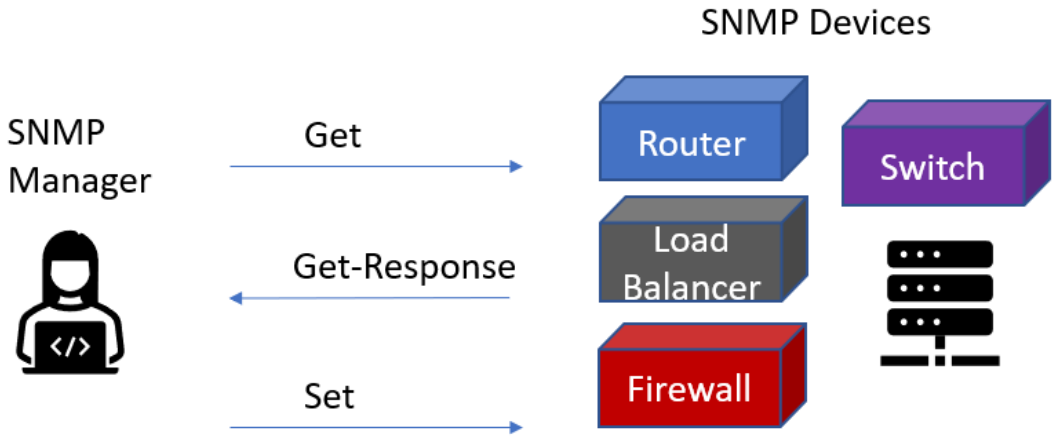

Figure 5. Simple Network Management Protocol

Each device maintains a Management Information Base, which is a standardised, hierarchical, data structure where device attributes and performance metrics are tracked.

Essentially a network manager can configure and monitor all the devices in their network by issuing simple SNMP 'get' and 'set' commands and tracking responses. This model has significantly helped to transform and simplify the automated management of the vast collection of local area networks which span the internet.

F. Process Algebra

A process algebra is a mathematical model of processes, that act and interact continuously with each other and with their common environment. One of the most successful process algebras is CSP (Communicating Sequential Processes) which was invented by C. A. R. Hoare and then developed further in partnership with A. W. Roscoe and S. D. Brookes [1,2].

CSP has been instrumental in enabling deep understanding and analysis of complex concurrent systems, through abstraction, by providing a structured and rigorous approach to reasoning about concurrency.

Processes are defined using events, operations and parallel combinations of other processes. Abstraction is provided through the use of *non-determinism*: behaviours which are irrelevant to the

purpose of creating an abstract model of a concurrent system are represented as unpredictable internal decisions.

The basic syntax of CSP processes and operations is described by the following grammar

$$
\begin{array}{lll}
Process & :== & STOP \mid \quad \text{Deadlock} \\
 & & SKIP \mid \quad \text{Termination} \\
 & & event \rightarrow Process \mid \quad \text{Event prefix} \\
 & & Process \,;\, Process \mid \quad \text{Sequential join} \\
 & & Process \;[\![alph \mid alph]\!]\; Process \mid \quad \text{Parallel join} \\
 & & Process \;|||\; Process \mid \quad \text{Interleave} \\
 & & Process \sqcap Process \mid \quad \text{Choice (internal)} \\
 & & Process \,\Box\, Process \mid \quad \text{Choice (external)} \\
 & & Process \setminus event \mid \quad \text{Event renaming} \\
 & & f(Process) \mid \quad \text{Named process} \\
 & & name \mid \quad \text{Recursion} \\
 & & \mu\, name \bullet Process
\end{array}
$$

Figure 6. CSP Syntax

This notation allows for a precise and unambiguous description of concurrent systems. Processes which are defined using this language obey a set of algebraic rules which enables formal mathematical reasoning to be conducted concerning their behaviour [2, 20].

$$
\begin{aligned}
SKIP \,;\, P &= P \,;\, SKIP = P \\
STOP \,;\, P &= STOP \\
(P \,;\, Q) \,;\, R &= P \,;\, (Q \,;\, R) \\
(a \rightarrow P) \,;\, Q &= a \rightarrow (P \,;\, Q) \\
P \;[\![A \mid B]\!]\; Q &= Q \;[\![B \mid A]\!]\; P \\
P \;[\![A \mid B \cup C]\!]\; (Q \;[\![B \mid C]\!]\; R) &= (P \;[\![A \mid B]\!]\; Q) \;[\![A \cup B \mid C]\!]\; R \\
P \;|||\; Q &= Q \;|||\; P \\
P \;|||\; SKIP &= P \\
P \;|||\; (Q \;|||\; R) &= (P \;|||\; Q) \;|||\; R \\
P \sqcap P &= P \\
P \sqcap Q &= Q \sqcap P \\
P \sqcap (Q \sqcap R) &= (P \sqcap Q) \sqcap R \\
P \,\Box\, P &= P \\
P \,\Box\, Q &= Q \,\Box\, P \\
P \,\Box\, (Q \,\Box\, R) &= (P \,\Box\, Q) \,\Box\, R \\
P \;[\![A \mid B]\!]\; (Q \sqcap R) &= (P \;[\![A \mid B]\!]\; Q) \sqcap (P \;[\![A \mid B]\!]\; R)
\end{aligned}
$$

Figure 7. Some Laws of CSP

These algebraic laws of CSP can be used to prove properties of arbitrary CSP networks. Fortunately there also exists a mature and established automated proof tool for checking properties of finite-state networks. This is FDR (Failures Divergences Refinement) [8], which can prove that a CSP process network implements a specification, also modelled in CSP, using the principle of refinement. In CSP, a system is said to refine a specification if its behaviours are a subset of the specification system. Hybrid proofs using a combination of algebra and automation can also be very effective [21].

At the COPA2021 conference I presented a case study of a prototype insurance claims automated payment system, modelled as a set of CSP parallel processes [9] (see figure 8). The system had been observed to contain a bug which allowed payment of a claim to erroneously repeated. Having coded up an abstract representation of the system and the requirement for a claim to be paid no more than once, I ran a refinement check in FDR which reported a counter example and provided me with a possible sequence of events that could lead to a double payment. I was then able to modify the logic of the implementation to remove the bug, and also to verify this new design with FDR.

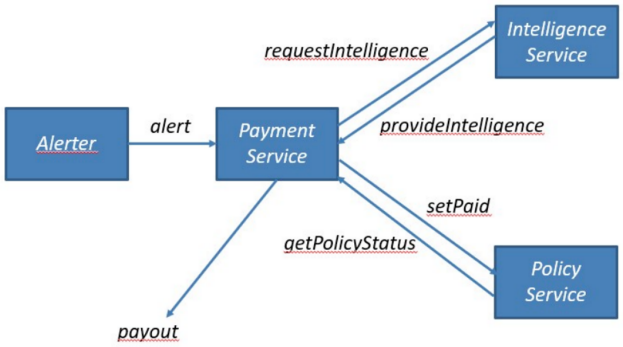

Figure 8 - Microservices for Insurance Claims System

This was a simple example where the FDR engine was able to formally verify the existence of a bug in a realistic concurrent system, making use of abstraction to hone in on the problematic aspects of the solution.

CSP has been phenomenally successful as an abstract model for driving forward both theoretical results and also practical results of immense significance, such as cracking the famous Needham-Schroeder cryptographic protocol [22].

```
SPEC = STOP ⎕ payout.Pol1 → STOP

CHECK'(pol) =
requestIntelligence.pol→provideIntelligence!pol?r→
getPolicyStatus.pol?s →
if r == Total and s == Unpaid then
  payout.pol → SKIP
else
  SKIP

MESSAGEBUS = payout?pol → setPaid.pol → MESSAGEBUS

PAYMENTSYSTEM' =
alert → CHECK'(Pol1); CHECK'(Pol2); PAYMENTSYSTEM'
```

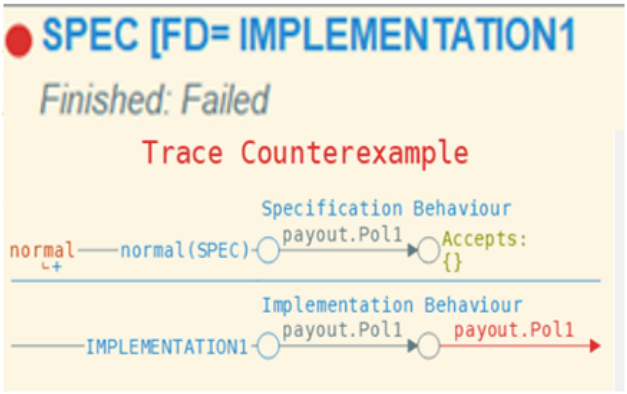

Figure 9. Output from FDR model checker

III. More Recent Concurrency Abstractions

A. Cloud Computing

One of the most significant steps in computer evolution over the past decade, has been the wholesale adoption of Cloud Computing [13, 23].

An abstract model for cloud computing that hides both physical infrastructure and geographical location can be described using several layers or components, as shown in figure 10. This model compartmentalises the underlying hardware, the data centres, and their physical locations, in lower levels – hence allowing the users to focus solely on their computing needs in the presentation layer.

Cloud managers can easily construct and maintain *virtual* data centres for their organisations, installing new servers and networking devices at the click of a button.

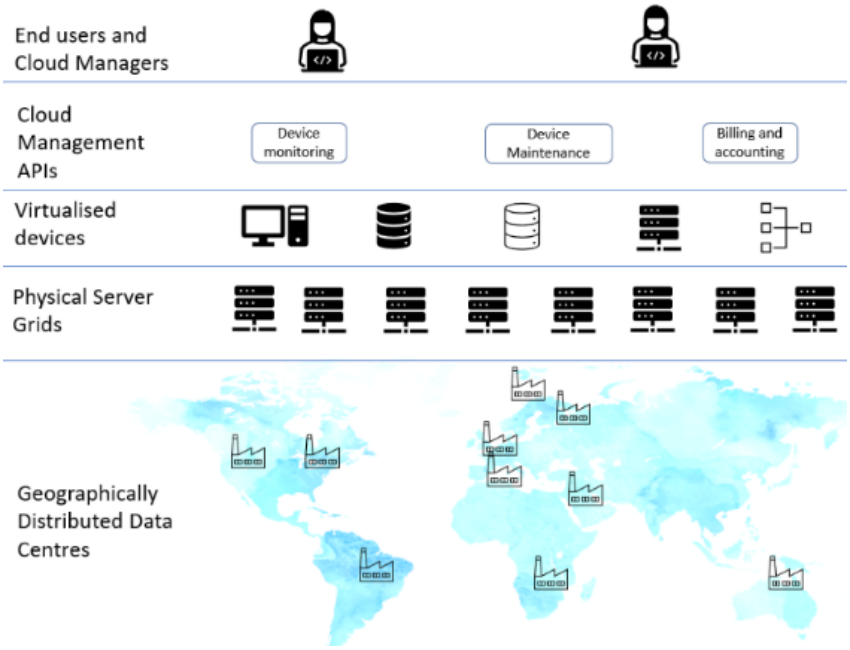

Figure 10. Abstract cloud computing model

The emergence of cloud computing has hugely facilitated business agility and innovation. And it has stimulated the important emergence of the DevOps movement - where organisations can combine their development and operations teams, with infrastructure and information security requirements being defined as code.

B. Google Map-Reduce

The launch of Google's search engine had a seismic impact on the internet, technology, and the way people access information for both work and leisure. It offered unprecedented speed, efficiency and quality of results. At its heart was an abstract design pattern called Map-Reduce, named after two functional programming paradigms [5].

Google's web crawlers continuously gather and index web pages from the internet. Although the full details are proprietary, it is believed that their process runs as follows. The collected pages are stored in a distributed file system and divided into smaller units for parallel processing which are distributed to multiple worker processes. The 'Map' function is then performed, whereby a user-defined function processes each shard of data in parallel, extracting relevant information from the web pages, such as keywords, links, and metadata. The results are emitted as intermediate files containing key-value pairs. These intermediate files are then passed to another set of worker processes which apply the 'Reduce' function to consolidate the final results into a condensed form that can be used to answer search queries very efficiently.

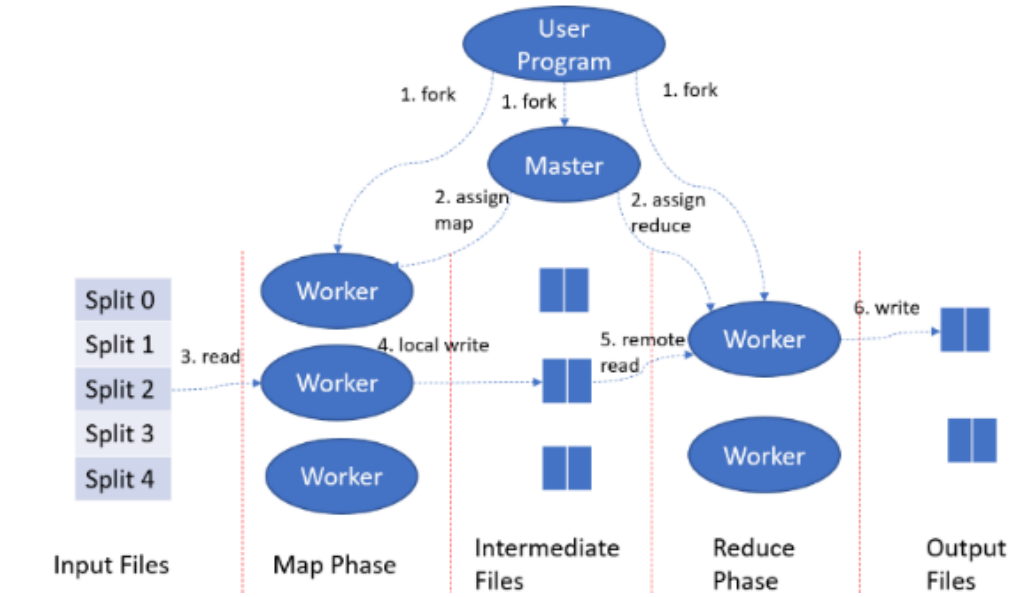

Figure 11. Basic MapReduce pattern

C. Data Lake

The Google Map-Reduce abstraction has subsequently been adopted by the Data Science community for so-called 'Big Data Analysis' and is at the heart of the Data Lake architectural abstraction [15] model for storing and managing vast amounts of diverse data in a centralized repository

Data Lake is intended to provide organizations with a flexible and scalable solution for storing, processing, and analysing both structured and unstructured data.

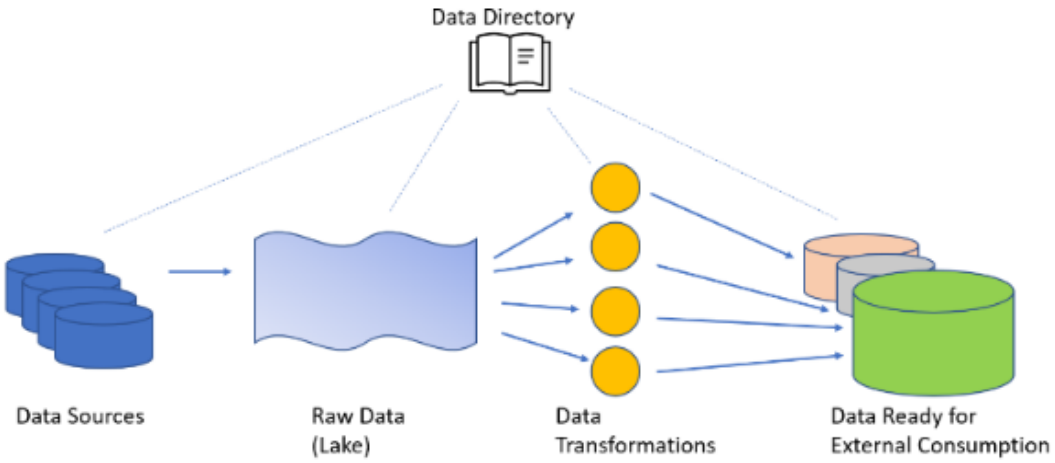

Figure 12. The Data Lake

A data lake ingests external data sources into a storage area, known as the Raw Zone. The data typically arrive from a wide variety of sources, such as traditional databases, IoT devices, social media, and surveillance videos. The data pass through transformation steps, to be cleaned, analysed and joined as appropriate. Ultimately they arrive in output data stores, ready for external data consumption. 'Big Data' analytic tools, such as Hadoop and Spark, which are based on Map-Reduce, are used to perform the data transformations on grids of parallel computers.

A key element of the pattern is the Data Directory, which maintains metadata related to every data set that passes through the lake: type, lineage, ownership, access rights, quality, and adherence to policies such as GDPR.

Adoption of this simple pattern is transforming data management and governance of many enterprises which previously struggled to maintain their data in complex collections of aging data siloes.

D. Microservices – Chaos Monkey

Microservices Architecture [6, 7, 9] is a modern flavour of Communicating Process Architecture, based on fine-grained services and lightweight protocols. It represents a fundamental shift in Solution Delivery practice away from building complex, multi-tiered monoliths. Its main principles are as follows:

- A microservices-based IT system is delivered as a set of loosely coupled components;
- Each component service implements one specific capability from an enterprise perspective;
- Components can be independently developed, potentially harnessing different technologies as appropriate;
- Communication uses technology agnostic protocols (which means you can design them independent of infrastructure needs to run anywhere);
- Microservices are small in size so that they are suitable for implementation and support by a DevOps team with a continuous delivery software development mind-set.

Microservices are used for replacing the individual monolithic services with collections of simple, single-purpose, lightweight processes and are highly concurrent. There might be many instances running of each microservice, from different locations, with the numbers scaled up and down on demand.

One of the early adopters of Microservices was Netflix for their video streaming service which was implemented using the AWS Cloud. They took a strategic decision to develop a culture among their engineers of building redundancy and automation into the system to make it resilient to AWS outages without any impact to the millions of Netflix members around the world. They did this by introducing the *Chaos Monkey* which randomly chooses servers in their production environment and turns them off during business hours.

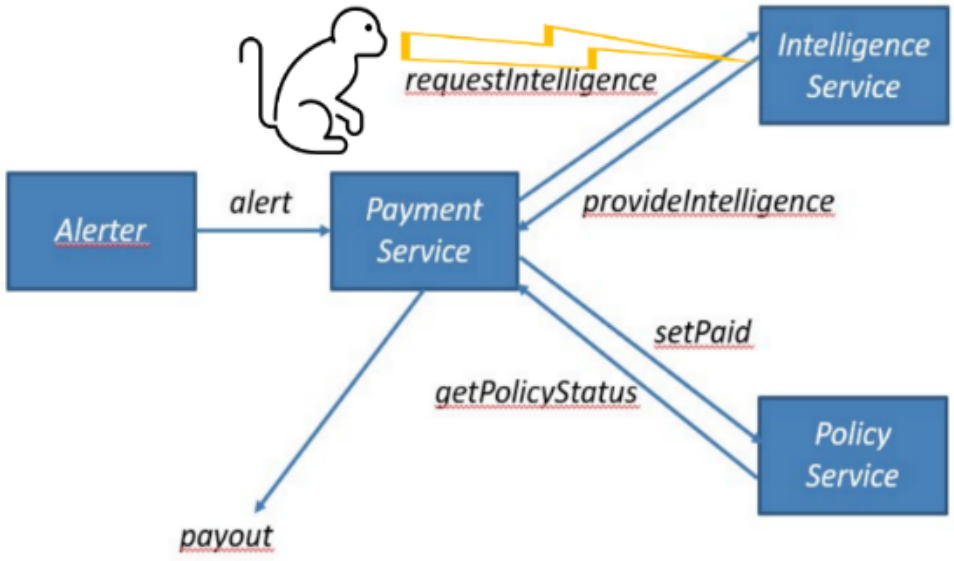

Figure 13. Microservices Architecture Pattern

The Microservices and the Chaos Monkey abstractions work together to help organizations build and maintain resilient highly concurrent and distributed systems. Production microservices are periodically deliberately caused to fail, providing invaluable data and insights to the product team, enabling them to ensure that their services are resilient to unexpected disruptions, leading to a more reliable system.

IV. Conclusions and Potential Future Abstractions

I have presented some significant concurrency abstractions from the past sixty years and illustrated how they have facilitated major technological advances to take place which have transformed certain aspects of how we live. These advances have facilitated major improvements to the general efficiency and prosperity of the modern world. (But it should be said that there is also a downside to some of these.)

I shall now describe two currently challenging areas of technical research where new concurrency abstractions are needed to help drive them forward.

A. Quantum Computing Simple Language

Quantum computing [16] is a rapidly evolving field that leverages the principles of quantum mechanics to perform computations in a fundamentally different way than classical computers. They were originally proposed in 1984 independently by Feynman [25] and Manin. At the time of writing, small-scale quantum computers have become a reality and are available for hire from cloud service providers. However they are not yet sufficiently powerful to carry out useful computations, they are still primarily a research tool.

Theoretical work has shown that certain calculations have a lower quantum complexity than computational complexity. For instance a quantum computer could potentially carry out integer factorisation exponentially faster than a traditional computer, make it feasible to break many of the cryptographic systems in use today, with potentially devastating consequences for global commerce and national security.

But there is also a school of thought that it might never be possible to create sufficiently powerful quantum computers to realise this potential because of problems with managing increasing error-rates as the number of components increases.

Quantum computers are analogous to classical computers in that they consist of circuits with gates that process bits of information. In classical computing a bit can only take the value 0 or 1, but in Quantum Computing the *qubit* can exist in multiple states simultaneously, due to the property of *superposition*.

These computers also exhibit the strange phenomenon of *entanglement* between qubits. Their states can become linked together so that the measurement of one instantly affects the state of the other.

Quantum gates are used to process the qubits. These are analogous to classical logic gates but may perform more complex operations, taking advantage of superposition and entanglement.
Whereas the gates of a digital circuit perform simple Boolean operations, like AND, OR, and NOT, on classical bits, which can only take the values 0 or 1, quantum circuits require a more advanced level of mathematics to describe. Qubits are unit length complex vectors in a Hilbert vector space, and may take an infinite number of possible values. Quantum gates perform operations on one or more qubits by applying a tensor product between the gate's matrix representation and the state vector(s) of the qubits.

The mathematical properties of superposition and entanglement can be used to design highly efficient quantum algorithms for solving important problems, potentially processing vast amounts of information in parallel.

However, Quantum Computing, like Quantum Mechanics, is hard to grasp intuitively because of the strangeness of the concepts of superposition and entanglement. This makes the entry bar seem rather high for becoming a quantum developer.

But there was a time when programming a classical computer would have required a significant understanding of electronic circuits and components. And we have moved a very long way from that in the past eighty years with a stack of abstractions for digital circuits, microprocessors, distributed memory protocols, machine code, assembly language, and high level programming languages.

Simulators exist for quantum circuits, using classical computing, with mathematical concepts, such as complex vectors, matrices and tensor products used to represent the entities and operations. One way for traditional programmers to understand Quantum Circuits as a kind of restricted programming language, where the only data type is qubit and the operators are defined as matrices. But, because of the nature of Quantum Physics, some of these operations can lead to more efficient execution than would be possible in a simulator.

Building abstraction models for quantum computing has become a rich area of research. Languages, such as Q#, Qiskit, or Cirq, provide high-level abstractions for programming quantum algorithms. Programmers can express quantum operations and algorithms using familiar syntax. Quantum Libraries are available for these languages providing pre-built quantum algorithms and circuits that programmers can use as building blocks. This is essentially at the same level as assembly language for classical computing, and further layers of abstraction and simplicity are needed to drive progress.

So the race is on to come up with an abstract model for quantum computing, that makes the field more accessible to conventional programmers without requiring them to have an advanced understanding of Quantum Physics.

It would be interesting to explore whether use of a process algebra, such as CSP, could be instrumental in achieving this. After all, entanglement is a similar concept to synchronous communication, and circuits represent a process workflow.

B. Defence Against Cyber Terrorism

Cyber Terrorism is one of the major threats to peace and stability that the world now faces, part of the flip side of the many advantages that technical advances have brought us.

Lowe [22] and Roscoe [2] performed some seminal work using CSP and FDR to model a spy attempting to compromise many established cryptographic protocols, most of which they found ways to crack. In Roscoe's approach the spy was modelled as a process which can 'overhear' certain messages and accumulate facts which will enable them to fake certain new messages. The spy is able to fake messages based on any facts they have accumulated and will arbitrarily do so with no particular strategy. However the exhaustive nature of the validation carried out meant that the existence of an intelligent strategy that would crack the protocol would be revealed using the approach.

We now face a cyber terrorism threat where very significant resources are being dedicated to attempting to infiltrate many IT systems that are vital to our security and prosperity. These attacks can take place on many fronts using all relevant state-of-the-art technical capabilities available, such as AI, big data analytics, social media indoctrination [17], and also breaching physical security controls to gain direct access to critical systems.

In order to protect our society against these threats we need to develop abstract models to support the rapid development by our engineers and technical security experts of innovative and powerful terrorism defence tools.

Creating tools to protect against cyber terrorism requires a multi-faceted approach that involves technology, policies, and expertise from various domains.

Developing an abstraction model for such tools would involve simplifying and structuring the process of cybersecurity in a way that enables efficient development, deployment, and management of protective measures against cyber terrorism. This could become a natural successor to Lowe's CASPER language - Compiler for the Analysis of Security Protocols – which is based on CSP and FDR [26].

REFERENCES

[1] C.A.R. Hoare, “Communicating Sequential Processes,” Prentice-Hall, 1985.

[2] A.W. Roscoe, “The Theory and Practice of Concurrency,” Prentice-Hall, 1998.

[3] Andrew S. Tanenbaum and David J. Wetherall. “Computer Networks”, 5th Edition, Prentice-Hall, 2010.

[4] David A. Patterson, John L, Hennessey. “Computer Organization and Design: The Hardware/Software Interface”. 5th Edition, Morgan Kaufmann, 2013

[5] Jeffrey Dean and Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, Communications of the ACM, 2008

[6] Sam Newman, “Building Microservices: Designing Fine-Grained Systems,” O’Reilly, ISBN: 978149195 0357, 2014.

[7] Leonard Richardson and Sam Ruby, “RESTful Web Services,” O’Reilly, 2007.

[8] T. Gibson-Robinson, P. Armstrong, A, Boulgakov and A. W. Roscoe. “FDR3: a parallel refinement checker for CSP,” Int J Softw Tools Technol Transfer 18, 149–167 2016.

[9] Jeremy M. R. Martin. “Designing and Verifying Microservices Using CSP”. Proceedings of 2021 IEEE Concurrent Processes Architectures and Embedded Systems Virtual Conference, IEEE

[10] Henry F. Korth. “Locking Primitives in a Database System”, Journal of the Associatio12n for Computing Machinery, Vol 30, No 1, January 1983, pp 55-79.

[11] Jörg Keller, Christoph Kessler, Jesper Träff. ”Practical PRAM Programming”, J. Wiley & Sons, 2001

[12] JD Case, M Fedor, ML Schoffstall, J Davin. “A Simple Network Management Protocol (SNMP)”, Network Working Group Request for Comments: 1098 1990

[13] Brian Hayes. “Cloud Computing”, Communications of the ACM Volume 51, Number 7 (2008), Pages 9-11

[14] Michael Alan Chang, Brendan Tschaen, Theophilus Benson, Laurent Vanbever. “Chaos Monkey: Increasing SDN Reliability through Systematic Network Destruction”, SIGCOMM ’15 August 17-21, 2015, ACM

[15] Fatemeh Nargesian, Erkang Zhu, Renee J. Miller, Ken Q. Pu, Patricia C. Arocena. “Data Lake Management: Challenges and Opportunities”, Proceedings of the VLDB Endowment, Vol. 12, No. 12 ISSN 2150-8097 2019

[16] Robert Hundt, “Quantum Computing for Programmers”, Cambridge University Press 2022.

[17] Sinan Aral, Dean Eckles. “Protecting elections from social media manipulation”, Science Vol 365, Issue 6456. 2019

[18] James A. Lewis. “Assessing the Risks of Cyber Terrorism, Cyber War and Other Cyber Threats”, Center for Strategic and International Studies 2002.

[19] J. Martin and P. Welch. “A Design Strategy for Deadlock-Free Concurrent Systems”, Transputer Communications Volume 3 Number 4 (1996)

[20] J. Martin, “The Design and Construction of Deadlock-Free Concurrent Systems”, D. Phil Thesis (1996), University of Buckingham

[21] P. H. Welch and J. M. R. Martin *“A CSP Model for Java Multithreading”.* Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 2000), IEEE Press 2000

[22] Gavin Lowe. “Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR”, in Margaria, T., Steffen, B. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 1996. Lecture Notes in Computer Science, vol 1055. Springer, Berlin, Heidelberg 1996

[23] Jeremy M. R. Martin, Steven J. Barrett, Simon J. Thornber, Silviu-Alin Bacanu, Dale Dunlap, Steve Weston. “Economics of Cloud Computing : a Statistical Genetics Case Study”, Proceedings of Communicating Process Architecture 2009, IOS Press 2009

[24] C. Bennett and A. Tseitlin. “Netflix: Chaos monkey released into the wild. netflix tech blog” 2012

[25] Feynman, Richard. "Simulating Physics with Computers", International Journal of Theoretical Physics. 21 (6/7): 467–488 1982

[26] Gavin Lowe, “Casper: A Compiler for the Analysis of Security Protocols”, https://www.cs.ox.ac.uk/gavin.lowe/Security/Casper/