ACM SIGACT News Distributed Computing Column 21

Sergio Rajsbaum*

Abstract

The Distributed Computing Column covers the theory of systems that are composed of a number of interacting computing elements. These include problems of communication and networking, databases, distributed shared memory, multiprocessor architectures, operating systems, verification, Internet, and the Web. This issue consists of:

- "Travelling through Wormholes: a new look at Distributed Systems Models," by Paulo E. Veríssimo.

Many thanks to Paulo for his contribution to this issue.

Request for Collaborations: Please send me any suggestions for material I should be including in this column, including news and communications, open problems, and authors willing to write a guest column or to review an event related to theory of distributed computing.

Travelling through Wormholes: a new look at Distributed Systems Models

Paulo E. Veríssimo [1]

Abstract

The evolution of distributed computing and applications has put new challenges on models, architectures and systems. To name just one, 'reconciling uncertainty with predictability' is required by today's simultaneous pressure on increasing the quality of service of applications, and on degrading the assurance given by the infrastructure. This challenge can be mapped onto more than one facet, such as time or security or others. In this paper we explore the time facet, reviewing past and present of distributed systems models, and making the case for the use of hybrid (vs. homogeneous) models, as a key to overcoming some of the difficulties faced when asynchronous models (uncertainty) meet timing specifications (predictability). The Wormholes paradigm is described as the first experiment with hybrid distributed systems models.

*Instituto de Matemáticas, UNAM. Ciudad Universitaria, Mexico City, D.F. 04510. rajsbaum@math.unam.mx.

[1]pjv@di.fc.ul.pt. Univ. of Lisboa Faculty of Sciences. Bloco C6.3.10, Campo Grande, 1749-016 Lisboa, Portugal. Tel. +(351) 21 750 0103. Fax +(351) 21 750 0533. Navigators Home Page: http://www.navigators.di.fc.ul.pt. This work was partially supported by the FCT, through the Large-Scale Informatic Systems Laboratory (LaSIGE), and by the EC, through IST projs. RESIST and CRUTIAL.

1 Introduction

The evolution of distributed computing and applications has put new challenges on models, architectures and systems. At the same time, evolution of technology has brought new opportunities that should not be wasted by researchers on distributed systems [28, 26].

To name just one grand challenge, 'reconciling uncertainty with predictability' is required by today's confluence of two conflicting goals [29]. On one side, the pressure on increasing the quality of service of applications as seen by users (response, determinism, robustness, security). On the other side, the continued lack of guarantees given by the infrastructure, because dramatic improvements in technology peer with the increase in asymmetry and instability brought by access networks, mobility, de-regulation, etc. To exemplify the magnitude of the challenge, it can be mapped onto at least two very important facets:

- *Time facet*, where the asynchronous computing model unfortunately does not serve timed requirements, despite the fact that there is today practically no useful application where time is absent from, be it in the explicit form of timeliness properties, or in an implicit form as an artefact to achieve higher-level system properties (e.g., timeouts). However, the technology improvements in networks and computers have by no means put us in the position of being able to use the synchronous computing model in an unrestricted manner. We have had to learn how to overcome this contradiction: a few steps were given in the past few years and maybe we start seeing a pale light at the end of the tunnel.

- *Security facet*, where the Trusted Computing Base fault model, a.k.a. crash fault model in the security world (by designed-in intrusion prevention) has proved not to be realistic as a generic model for the construction of secure systems. On the other hand, the arbitrary, a.k.a. Byzantine fault model, a proven pessimistic abstraction for safety-critical applications subject to independent accidental faults, suffers from the difficulty of stochastically assessing the distribution of malicious faults (attacks and intrusions) perpetrated by hackers, let alone from the more than obvious risk of the lack of independence (common-mode faults).

Given this scenario, one can legitimately ask a few constructively sceptical questions:

- Should we continue to use asynchronous distributed computing models? And if yes, shouldn't we try to formalise their limitations under the constraints posed by these new scenarios?

- Should we continue to use Byzantine fault models and algorithms to address security-related problems? And if yes, shouldn't we try to formalise their limitations under these new constraints?

- If no, does the answer mean jumping right into the opposites of the above, crash and synchronous models? Or should we paraphrase the old saying 'the truth lies in the middle'?

In fact, take the time facet: should we talk about two models, synchronous and asynchronous? Or should we try and find a model capable of describing, perhaps in a parametric way, the spectrum of synchrony of a system, sweeping between the two extremes, pretty much like the spectrographic analysis of a chemical substance? Will not this be a better explanation of the world, after all the aim of any theory? Continuing the physics metaphor, such a model would still represent classical homogeneous models (when a single stripe occurs in the spectrum), but it would also open avenues for challenging new ways, like hybrid models (when more than one stripe occur in the spectrum).

Suppose that such hybrid abstract distributed system models could find a mapping onto (correspondingly hybrid) architectural models that reflect reality (the networking and computational environment) at least as well as the current ones. That is, we would be as comfortable with their faithfulness as we were, until today, with saying that the Internet is faithfully represented by the asynchronous model. Then, under these more

expressive models we might devise computational paradigms able to solve the contradictions expressed in the opening.

Some of these questions were equated over the past few years, and were addressed by a few people researching in this area. In this paper, we will survey some of what was done, trying to motivate problems and solutions. Using time/synchrony as an anchor theme, we show how hybrid models can be a powerful solution to some problems that push the asynchronous model to the limit, and an enabler for the construction of innovative algorithms. We present the Wormholes hybrid distributed system model, and discuss the most relevant issues concerning it.

In the security facet there are also very interesting results on very efficient and resilient Byzantine wormhole-aware algorithms, but for space reasons they will not be addressed in this paper.

2 The case for hybrid distributed system models

The FLP impossibility result set the stage for the difficulties to be encountered when wanting to solve problems related to time (the alma mater of synchrony) in essentially time-free systems: Fischer, Lynch and Paterson showed that any consensus protocol for asynchronous systems has the possibility of non-termination if a single process is allowed to crash [15]. The result extended by equivalence to another important primitive, atomic broadcast [7]. Several authors tried to overcome these difficulties in a number of ways, generally around these two important but very specific problems. See also [19, 25] for surveys.

What is the minimal synchrony to solve increasingly more problems in the time domain?

In the past, basically two solutions have been employed to circumvent this result. The first resorts to the use of randomisation [3], and the second to extending the basic asynchronous model with some time-related assumptions. These assumptions can be made explicitly [12, 13], or they can be encapsulated in some construct such as an unreliable failure detector [7]. Dwork et al. studied a range of restrictions to the fully asynchronous model that would enable the solution of consensus. Chandra & Toueg proposed the failure detectors, a very elegant way to structure consensus-related algorithmics, but in a very similar manner they gave a hierarchy of such detectors, requiring the system to be or become synchronous enough, and for long enough periods, in order to solve consensus [7].

Cristian & Fetzer devised the timed-asynchronous model, where the system alternates between synchronous and asynchronous behaviour [8], making progress when the system has just enough synchronism to make decisions such as 'detection of timing failures'.

Where should that synchrony be?

In [30] the following observation was made: *synchronism is not an invariant property of systems*. On the one hand, it was meant that the degree of synchronism varies in the time dimension: during the timeline of their execution, systems become faster or slower, actions have greater or smaller bounds. The works just cited have indeed relied on this variation as a problem solver. On the other hand, it was meant that it varies with the part of the system being considered, that is, in the space dimension: some components are more predictable and/or faster than others, actions performed in or amongst the former have better defined and/or smaller bounds. This was the innovative perspective at that time, and the insight that led to the Wormholes hybrid distributed system model reported in these pages.

Now, which of the two dimensions (time or space) is more constructive? That is, which dimension allows us to think about algorithms in the abstract drawing board which later lead to more realistic, feasible, resilient systems? There is no silver bullet, but a remarkable difference is that under the time dimension one *expects* the system to become adequately synchronous, whereas by exploring the space dimension i.e., acting on the system structure, one *makes* the necessary synchronism happen. That is, one can make some parts of the system exhibit a well-defined (and perpetual if desired) time-domain behaviour, regardless of the asynchronism of the rest of the system.

All the works cited above considered the eventual evolution of the system through periods of sufficient synchrony, during its execution: they only explored the time dimension. In essence, this explains why we vow for *hybrid distributed systems models*, where different loci of the system have different properties and can rely on different sets of assumptions (e.g., faults, synchronism). These models allow us to take the best from both dimensions, both in theoretical and practical terms, as we exemplify during this paper.

What is there to be gained vs. homogeneous models?

Hybrid distributed systems models are:

- *Expressive models w.r.t. reality*— Real systems are partially synchronous in the time dimension. But further to that, they generally have components with different degrees of synchronism, i.e., in the space dimension: different bounds or absence thereof, different speeds, different tightness and steadiness (metrics of synchronism, see [32]). Homogeneous models simply cannot take advantage from this, being confined to use the worst-case values or bounds (e.g., of the least synchronous component), which ultimately— and safely— mean asynchrony.
- *Sound theoretical basis for crystal-clear proofs of correctness*— Why were some problems found in earlier-generation asynchronous algorithms [6, 2]? One of the reasons was because timing assumptions were made for the system that were not in agreement with the model. Artificially restricting such assumptions to 'parts' of an asynchronous system does not improve the situation much if it follows a homogeneous model: we may fall into the same kind of contradictions. However, by using a hybrid model, the heterogeneous properties of the different loci of the system (the space dimension...) are by nature represented, and we are in consequence forced to explicitly make correctness assertions about each of these loci, and about the interfaces to one another.
- *Naturally supported by hybrid architectures*— Sisters to hybrid distributed systems models, hybrid architectures accommodate the existence of actual components or subsystems possessing different properties than the rest of the system. Hybrid models and architectures provide feasibility conditions for powerful abstractions which are to a large extent unimplementable on canonical (homogeneous) asynchronous models: failure detectors; ad-hoc synchronous channels; timely execution triggers (a.k.a. watchdogs) or timely executed actions (periodic or event-triggered) [2]. Hybrid models and architectures may drastically increase the usefulness and applicability of all these abstractions.
- *Enablers of concepts for building totally new algorithms*— A powerful yet simple concept behind the first experiments with hybrid models was: use the weakest possible model for the generic system; imagine that a "toolbox" of simple but stronger low-level services is available, locally accessible to processes (e.g., timely execution triggers; timely executed actions; trusted store); these local services can be distributed via alternative channels, to obtain further strength (e.g. synchronous channels; trusted global time; trusted binary agreement); devise algorithms which, by working between the two space-time realms, the generic and the enhanced subsystem containing the "toolbox", achieve new properties (e.g., making an asynchronous process enjoy timely execution).

Hoping to have successfully motivated the problem, we are going to present, in a brief and necessarily informal way, the hybrid distributed systems model, nick-named *Wormholes model* after the astrophysics theory, and a companion architecting concept, called *architectural hybridisation*.

3 The Wormholes model

We introduce the Wormholes model with the help of a metaphor:

> The fastest speed in the universe is the speed of light, which would make it impractical to

[2] The same could be said of security-related abstractions concerning tamper-proofness vs. Byzantine models.

travel to, or communicate with, remote parts of the universe. However, a theory argues that one could take shortcuts, through, say, another dimension, and re-emerge safely at the desired point, apparently much faster than what is allowed by the speed of light. Those shortcuts received the inspiring name of *Wormholes* [3]. In essence, Wormholes prefigure an ancillary theory which coexists with the classical theory, predicting subsystems which present exceptional properties allowing to overcome fundamental limitations of the systems under the classical theory.

A hybrid distributed system model features several subsystems following different sets of assumptions, e.g. about synchrony or faults. In theory, nothing prevents a hybrid model where, for example, several synchronous subsystems coexist with several asynchronous ones. However, note that the instance of such model meeting the best practice of using the simplest possible model with the weakest possible assumptions is the one that fulfils the metaphor: a weak main or 'payload' subsystem; a few small, simple wormholes.

Moving from metaphor to reality, let us define the *Wormholes distributed system model* . For the sake of simplicity and without loss of generality, we assume a bi-modal system, with one payload system, and one wormhole subsystem, more complex systems can be recursively defined:

- There is a *payload system* S_p where algorithms and applications normally execute, composed of N_p payload processes p_i that communicate via message passing, through *payload channels*.
- S_p follows a set of fault and synchrony assumptions H_p (normally weak, such as processing and communication being asynchronous, and faulty behaviour Byzantine).
- There is a *wormhole subsystem* S_w composed of N_w wormhole processes w_i. Wormhole processes may or not communicate amongst themselves, in which case they do so via message passing, through *wormhole channels*.
- S_w follows a set of fault and synchrony assumptions H_w normally stronger than the payload (such as processing and communication being synchronous, and faulty behaviour crash [4]).
- The only way for payload processes to communicate with wormhole processes is through wormhole gateways, W_G, with well-defined interfaces. The specific type of interface is not part of the model.
- Likewise, for payload processes the properties offered by any wormhole are defined and enjoyed *at* a wormhole gateway. The payload processes do not have to know how wormholes are implemented, and vice-versa.
- The relative number of payload and wormhole processes is not part of the model. In fact, maybe not all payload processes access wormholes in certain algorithms, or maybe more than one payload process can access a same wormhole.

In practical terms, a wormhole is a privileged artefact to be used only when needed, and supposedly implements functionality hard to achieve on the payload system, which in turn should run most of the computing and communications activity.

Note that the payload and the wormholes follow different sets of assumptions, but there is no pre-assumption about what these sets are. For example, the payload may be asynchronous and/or have Byzantine failures, but it may also have some synchrony for a start (imagine wormhole-aware real-time systems...). Likewise, a wormhole may be synchronous, partially synchronous, etc. as fits the needs of the problems to solve.

So, in summary, the key innovative characteristic of the Wormholes model consists in making some stronger properties (e.g., synchrony) happen in a well-defined and safe way, whilst preserving the canonical model's weak abstractions (e.g., asynchrony). Note that whilst the most fascinating and powerful incarnation

[3] The interested reader may refer to URL: http://www.wordiq.com/definition/Wormhole

[4] This is the extreme, for the sake of example, but we hope to have left clear that wormholes can assume any weaker synchrony or fault model.

of a wormhole would be distributed (through alternative channels with enough synchrony), the simpler versions, local wormholes, still provide very useful support (e.g., local security and/or timeliness functions).

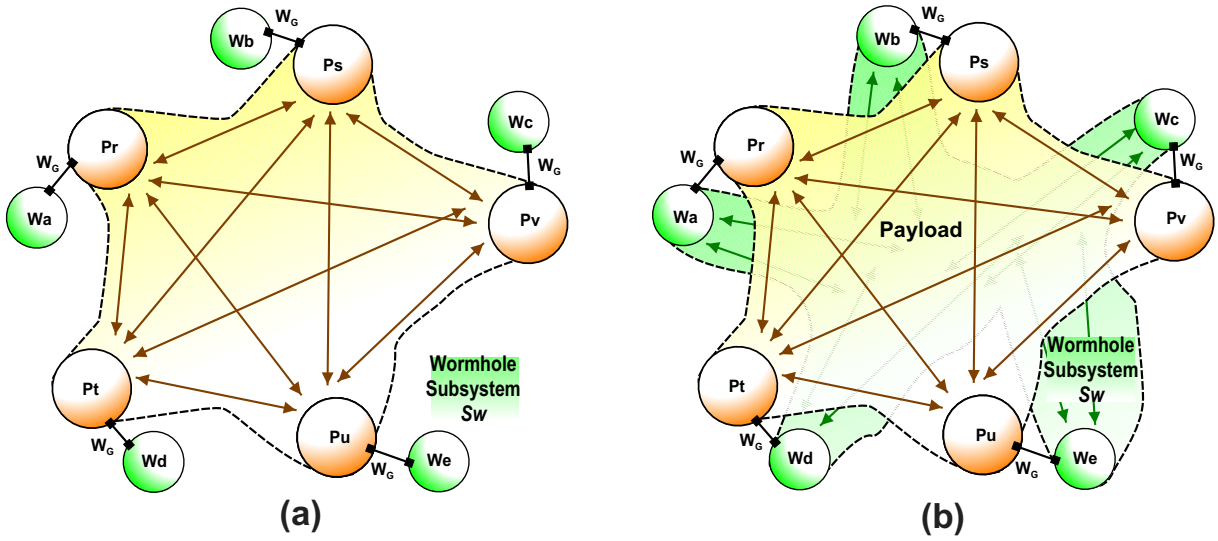

Figure 1: The Wormholes model: (a) Local wormholes; (b) Distributed wormholes

Figure 1 gives a picture of the model just presented, in a bi-modal incarnation. Figure 1(a) suggests a model with local wormholes, whereas Figure 1b depicts a distributed counterpart. The local wormholes case is the special case of this model where wormhole processes do not communicate directly with one another.

This model ensures a clear separation of concerns between the properties offered by wormholes and their construction, and between the execution on either side of a wormhole gateway. For instance, consider a system with an asynchronous payload and timely execution wormholes, which guarantee that a function f invoked at their interface is always executed in a known bounded time. It is easy to reason about the invocation of f on a wormhole w_u, by a process p_k. Using whatever invocation method stipulated, the timing of p_k getting to invoke f at the interface concerns the payload system realm. The timing of w_u getting to execute f since invoked concerns the wormhole subsystem realm. Likewise if a possible result is expected to be returned by f: the time from invocation to return at the interface is what gets bounded in the wormhole realm, whereas the timing for p_k to *use* the result pertains to the payload realm. To give an intuition about reality, in this example one might imagine each p_i co-located with a local wormhole w_i implemented by some sort of real-time micro-kernel, accessed by p_i as a run-time system call. Although it seems counter-intuitive, with slightly more sophistication in the exemplified interface, asynchronous processes can indeed take advantage from synchronous executions. The reader is referred to [31, 22], where such techniques are described.

4 Architectural hybridisation

Is it possible to travel through Wormholes? This metaphoric question translates into two practical ones: Is it feasible to construct systems such as postulated above? Are systems with Wormholes of any real use?

Fortunately, it is easier to answer these questions than to travel in hyperspace. Recapitulating the observations made in the introduction, hybrid modelling of distributed systems is the path to achieving incrementally stronger behaviour taking the best of two worlds: retaining the canonical and useful weakest-assumptions paradigm of asynchronism; achieving synchronism in a predictable manner.

Architectural hybridisation was proposed as a new paradigm to architect modular systems, based on a few simple principles:

- Systems may have realms with different non-functional properties, such as synchronism, faulty behaviour, quality-of-service, etc.
- The properties of each realm are obtained by construction of the subsystem(s) therein.
- These subsystems have well-defined encapsulation and interfaces through which the former properties manifest themselves.

As to the construction, architectural hybridisation is an enabler of the construction of realistic hybrid distributed systems [29]. In fact, it is quite straightforward to build architecturally-hybrid systems. Some earlier systems already exhibited flavours of architectural hybridisation [24, 33]. More recently, a few experimental wormhole systems have been built supported by architectural hybridisation, and the feasibility of these implementations, both in the time and security facets, is amply discussed in [4, 10, 29]. A timeliness and security distributed wormhole implementation is downloadable from our page [5], for experimental work with distributed algorithms on hybrid systems.

As to the usefulness of wormholes, the reader is referred to [29] for an overview of the multiple potential uses of wormholes. In one of our experiments, we have prototyped a specific kind of wormhole subsystem called *Timely Computing Base (TCB)* that allows achieving timely actions in systems that can be asynchronous. At the TCB distributed wormhole gateway, a set of simple but extremely helpful services are provided: timely execution; duration measurement; timing failure detection. We introduced a technique to interface a synchronous subsystem from a time-free one, making the asynchronous system perform timely (synchronous) actions or detect the failure thereof [31]. In [30] we present a formal embodiment of the TCB model and architecture.

The power of a wormhole such as the Timely Computing Base is interesting: immersed in an asynchronous system, it makes feasible the construction of useful abstractions such as perfect failure detectors, triggers such as watchdogs, periodic task dispatchers, or even mere synchronous communication channels [18, 14, 5, 35, 17]. However, it is important to note that a TCB is *sufficient* to implement a number of interesting paradigms, but it is by no means *necessary* to implement all of them. The TCB, fully synchronous and distributed, was intended as a proof of concept of wormholes, and not *the* ultimate wormhole. If it proved, as it did, to be easy to implement such a component— and feasible even in large scale settings [29]— then doors would be open for weaker and simpler forms of wormholes: subsets of the TCB services, including just local services; raw services such as synchronous bare channels or trusted real-time clocks; partially synchronous variants; trusted variants for security purposes, etc.

In the malicious failure domain, resilience to intrusions (a.k.a. intrusion tolerance) can be drastically augmented by using wormholes. In another experiment, we devised a set of new functions resilient to malicious faults for this new wormhole, calling it Trusted Timely Computing Base [10], and showed ways to perform trusted actions in the presence of uncertain attacks and vulnerabilities, such as solving consensus quite efficiently [9, 22].

5 Travelling through Wormholes

A few questions were raised by several readers lately, about the use and the potential of the Wormholes model. It is generally accepted that wormhole implementations make interesting contributions by showing how several typically made theoretical assumptions such as synchrony or eventual synchrony, or paradigms like failure detection, can be substantiated in a real "asynchronous" system. However, the power of wormholes as hybrid distributed system models has further implications, some of which with theoretical reach.

[5] URL: http://www.navigators.di.fc.ul.pt/software/tcb/index.htm

The most important ones are discussed in this section.

5.1 The substance of assumptions

The issue of fault or synchrony assumptions deserves special attention. It is customary to say that one should make the weakest assumptions possible. While from the complexity and resilience viewpoints this is not arguable, let us take a slightly richer systems perspective:

> In computer science, assumptions should represent with sufficient precision the environment they are supposed to depict. For example, people assume that large-scale (i.e., Internet) systems are asynchronous not for the sake of it, but just because it is hard to substantiate that they behave synchronously. In other words, the confidence (also called *coverage*) on the former assumption is higher than on the latter one. In other words, the asynchrony assumption leads to safer designs in this case.

Note that under this viewpoint, the pair $\langle$*assumption;coverage*$\rangle$, and not just the assumption, is what measures its 'weakness' or 'strength'. This opens refreshing perspectives on system modelling. For example, in open systems the assumption 'A- A local function is executed in a known bounded time' can be made to have a significant coverage for sensible bounds. However, a similar assumption 'B- A message is delivered in a known bounded time' would have a significantly lower probability of being met. So although similar, A is weaker than B. We would have more confidence on the correctness of a system based on A than one based on B: the former system would be safer [23, 32].

If the reader is at least moderately convinced of these arguments, let us now take the hybrid model perspective. Suppose assumption A or B referred to a privileged part of an otherwise asynchronous system (respectively a real-time microkernel, and an alternative synchronous network): their coverage can be made *by construction* so high that either or both could be taken for granted.

On another tone, there is some important research on hybrid fault models [20, 34], that assumes different failure type distributions for system processes. In contrast with wormholes, these are not hybrid system models, but in fact, as the name implies, heterogeneous failure type distributions over a same homogeneous model. In an analogy with the cited partial synchrony works, there is a baseline weak fault assumption for processes (the weakest of the hybrid set, e.g., Byzantine) and an assumed restriction of the actual faulty behaviour of some of these Byzantine processes in one or more ways, e.g., omissions or even crash. In essence, these models, of stochastic inspiration, *expect* a given distribution of fault types to occur in a homogeneous system model, in which case more efficient algorithms can be given to solve important problems (such as Byzantine agreement, featured by most of these works). In consequence, the hybrid faults model is subject to the same potential issues of substance discussed above. However, the concept leads to quite effective algorithms and one could envisage removing these sources of concern by using architectural hybridisation to model and build systems with hybrid faults.

5.2 Wormholes and partial synchrony

Some may ask how different this is from a partially synchronous model where communication and processes have some synchronism, already studied in the literature[12, 13]. Some reasons make it different, as we discuss below, highlithing the contributions of the Wormholes model.

Firstly, since the model adopted by those studies is homogeneous, restrictions of asynchrony refer to *all* components (processes or channels). For example, if it is 'communication channels eventually become synchronous', then all channels must comply with this. In consequence, these restrictions are equated solely in the time dimension (that is, *expecting* eventual synchronism to occur). In the example above,

channels are expected to become synchronous without there existing any model feature substantiating these mutations from asynchrony to synchrony and back. These abstract assumptions can be given a great deal of substance under Wormhole models. For example, wormholes support the abstraction of components with heterogeneous properties, and as such, restrictions to asynchrony can be assumed for *some* components only, rather than for the whole system. This opens avenues for studying tighter theory results.

Secondly, one might still ask why could restrictions to asynchrony not be assumed just for some components in a classical homogeneous model. The problem is one of substance, as discussed in the previous section. In a homogeneous asynchronous model, there is little to substantiate such assumptions, and in consequence they are really very strong assumptions. Under the Wormholes model, any restrictions to asynchrony can be assumed also in the space dimension, materialised by the properties of a particular locus of the hybrid system model, that is, *making* synchronism occur, in an eventual or perpetual manner. For example considering, as done in some more recent works, that specific functions or routines are timely triggered, or executed, or specific messages have known delivery bounds— these are all hard to substantiate under a homogeneous asynchronous system model. However, the scenario would drastically improve if we brought to scene a hybrid model with wormholes of the adequate kinds resp.: a watchdog, a real-time dispatcher, or a few synchronous network channels.

Finally, whereas the problems studied under the partial synchrony umbrella are specific and have thus a semantics attached— of which the failure detector to solve consensus is an example— the wormholes offer generic solutions to any problem with any mix of synchrony.

5.3 Wormholes and FLP

Wormholes are another way to circumvent the FLP impossibility result. The first thing to notice is that a system with wormholes is a combination of asynchronous and non-asynchronous subsystems. Therefore, the FLP result can be circumvented. However, as we have explained earlier, this does not mean that it is equivalent to the partial synchrony combinations studied by Dwork et al. [12, 13]. The Wormholes model can represent all of the latter, but it also encompasses any combinations of hybrid behaviour of system components, where synchronous properties: are encapsulated in, and can be confined to, subsets of the components; and can for example be perpetual.

On the other hand, when talking about encapsulation, one might recall the work of Chandra & Toueg to circumvent FLP. The remarkable intuition in this work was the separation of concerns between what is time-free and what may not be, in a consensus algorithm. However, let us dissect the work of these authors: (i) they encapsulated a given semantics in an oracle (failure detector properties), such that this oracle would allow a given asynchronous algorithm to solve consensus; (ii) it so happened that fulfilling these semantics required synchronism; (iii) the difficulty of implementing failure detectors in an asynchronous system for solving consensus remained. And this is what we are going to discuss next.

5.4 Wormholes and Failure Detectors

One might think that a wormhole differs little from a failure detector: synchrony assumptions are encapsulated and hidden in a component. There is much more to this than meets the eye, as we explain next.

Failure detectors (FDs) are a very elegant concept from a theoretical point of view [7]. Using the well-known ‘oracle’ technique, the authors had the remarkable intuition of killing two birds with one stone: separating the main computation issues (consensus) from those related to the dynamics of the players involved in it (failure detection), abstracted by two intuitive properties, accuracy and completeness; and obliging the consensus structure to be completely time-free, which would logically confine to the oracle any computations that would one way or the other be related to time/synchrony, as predicted by the FLP result.

From a computational model's perspective, the FD oracle allows the construction of algorithms that can be completely asynchronous. From a system model's perspective, the one taken in this paper, an observation is in order: nothing is advanced, in the FD work, about the architecture or implementability of such detectors. In fact, the usual direct mapping of the FD computational model to the asynchronous system model unveils the fundamental contradiction of building some synchrony on top of a fully asynchronous environment, and this contradiction has emerged in several works in the past decade.

This contradiction is amplified by arguments about the advantage of having perfect failure detectors (the hardest to implement), given the applicability limitations of weaker ones [18]. Moreover, a recent paper has shown that "there is no free lunch" [11]: if we wish to do really useful things, in the presence of an unbounded number of failures (or uncertain, for the matter), we have to make correspondingly strong assumptions about our environment. In the cited paper, the authors argue about the need for perfect failure detectors (and no weaker).

None of this overshadows the importance of the algorithmic contribution of FDs. The wormholes just happen to yield solutions to the FD implementability problem: under the Wormholes distributed system model, the contradiction is removed. In fact, an asynchronous system with failure detectors is a system with asynchronous payload and a synchronous enough wormhole to support the chosen FD's assumptions. A consensus algorithm would run on the payload part, and the wormhole gateway would offer the FD interface.

But wormholes are more generic: the functionality in a wormhole is not predefined or confined to failure detection, and the problem to solve not confined to consensus. Instead, it is totally up to the algorithm creator, and as such this opens much more generic avenues than using FDs, which are only starting to be tracked. Stimulating results appeared recently, for example in the form of very efficient Byzantine-resilient consensus without failure detectors [9, 22].

5.5 Wormholes considered necessary

As pointed out in the introduction, the Wormholes model clarifies and represents many borderline situations where homogeneous asynchronous system models navigate the dangerous waters of timing assumptions. For example, when we use a watchdog to timely trigger an execution in an otherwise asynchronous system, we may not be aware that the only timely thing that happens is a *click* for the start of the execution. The execution itself is *asynchronous* and as such not time bounded by definition. One may still argue that if the watchdog caused a hardware reset, the reboot execution is guaranteed to be synchronous. In reality, maybe so. In theory, we have a problem: either we stick to the homogeneous asynchronous model, and that period of the execution is outside the boundary conditions for the model and thus not explained by it, which is a formally incorrect situation; or we use a hybrid model, which correctly represents the situation, by modelling as a wormhole both the watchdog and the low-level machinery that develop reboot until returning control to the (asynchronous) high-level world.

A second example is the use of local clocks in otherwise asynchronous systems, for example to trigger periodic actions. Again one may argue that it is quite realistic to rely on hardware clocks that run similarly to real time, deployed in any PC or workstation. Indeed, but again the only timely thing that happens are the hardware *ticks*, all the operating system post-processing to give users the `system_clock` is asynchronous by definition of the model. In reality we cannot expect the `system_clock` function of an asynchronous system to be a deterministic function of real time. In other words, we cannot guarantee any stronger semantics for such function than that of a sequence counter. One may still argue that at least down-timers might work (e.g., PCs have hardware devices that count down a number of time units to zero and give a timeout, used to mark periods). Note that this is similar to a watchdog, and all that was said in the previous paragraph applies. To give a simple intuition, imagine that the (asynchronous) reader decided to rely on

her real time alarm clock to wake her up in order to perform some action due at that time, but instead falls asleep again, wakes up two hours later, and executes the action: this would still be legitimate behaviour for an asynchronous entity, but we fear the action might have been hopelessly late...

In these situations where asynchronous models must be complemented with timing assumptions in order to address timeliness specifications, implicit or explicit, we have seen no alternative to the Wormholes model for a correct specification of such settings.

6 Conclusions

We have presented a new perspective to distributed systems modelling and architecting, by introducing the notion of hybridisation [30, 29]. In this paper, we discussed several advantages of using hybrid distributed systems models, or *Wormhole models*, backed by *architectural hybridisation*.

From an architectural viewpoint, hybridisation shows how the theory can be put to work in practice: by introducing feasibility conditions for the implementation of very important paradigms such as failure detectors, in essence non-implementable in classical asynchronous system models; and by bridging gaps between the possibility in theory and feasibility in practice, providing efficient building blocks for some theoretical solutions whose previous implementations were very complex.

The theoretical merits of the Wormholes model stand on more abstract ground. We showed that although from a distance, one could classify such model of partial synchrony, a closer look reveals the differences: classical partial synchrony works considered variations of asynchrony in a homogeneous model, and as such, they concerned the system as a whole; likewise, they only explored the time dimension, in the sense that these variations were assumed by expecting the eventual evolution of the system through periods of sufficient synchrony; finally, the problems studied under the partial synchrony umbrella were specific (e.g. consensus) and sometimes this makes respective solutions less devoted to address generic problems.

Wormholes bring new insights to distributed systems modelling and architecting:

- They allow extending partial synchrony to the space dimension— regardless of the asynchronism of the whole system, some parts of it exhibit a well-defined, and perpetual if desired, time-domain behaviour.
- They enforce the desired behaviour or "better" properties— this comes with architectural hybridisation, whereby certain components of the system have stronger behaviour by construction— leading to designs that are simultaneously more predictable and safer.
- They offer a generic framework for providing any such better properties— such components could host perfect failure detectors, inasmuch as they could host any other useful abstraction for building algorithms.

Once the intuition behind wormholes captured, as we hope to have achieved in this paper, it is quite easy to devise algorithms under the Wormholes model. The interested reader is referred to [31, 9, 22], which provide clear examples.

Facing the need for timeliness equated in the beginning, and given the difficulty of achieving even the slightest timed behaviour in asynchronous systems, several authors have introduced ad-hoc conditions or constructs to address this problem, which explicitly or implicitly point to, or prefigure, the concept of wormhole [18, 1, 16, 14, 5, 35, 21]. However, we hope to have shown that it is important to clearly follow a *hybrid* model from the beginning, if one is to achieve the complete potential of these constructs, and avoid potential problems deriving from implicit and non-substantiated timing assumptions. As such, we hope this paper will contribute to the advance of distributed systems modelling and algorithmics.

For space reasons, the security facet was not explored adequately in this paper, though there are at least as many challenging findings and evolutions to report, perhaps in a future paper, on the use of wormholes to build intrusion-tolerant (e.g., Byzantine-resilient) algorithms. Furthermore, an important recent finding points to the impossibility of building resilient asynchronous systems (ones that stay up and correct long enough to fulfil their mission) and to the high probability of this impossibility result causing the failure of such systems under malicious attacks [27]. This problem can be fixed through the use of wormholes.

Likewise, we almost exclusively discussed the merits of the new model in representing asynchronous systems and applications. As we said, asynchronous systems cannot possibly solve problems with timeliness specifications, and these are an increasing part of our everyday's computer-dependent life. This brings added importance to time in distributed systems, and in fact, there are interesting results about achieving timed (and even real-time) behaviour, by hybridisation of time-free algorithmics with timed wormholes, which for lack of space we could not address in this paper.

Acknowledgements

I wish to thank Sergio Rajsbaum for having challenged me to write this paper and patiently accepted my asynchrony in writing it. Interesting discussions on distributed systems models with Michel Raynal and Roy Friedman are warmly acknowledged. Along the past few years, several researchers of the Navigators team contributed to make the wormholes model evolve.

References

[1] M. Aguilera, G. Le Lann, and S. Toueg. On the impact of fast failure detectors on real-time fault-tolerant systems. In *Proc. of DISC 2002*, October 2002.

[2] E. Anceaume, B. Charron-Bost, P. Minet, and S. Toueg. On the formal specification of group membership services. Technical Report RR-2695, INRIA, Rocquencourt, France, November 1995.

[3] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30, August 1983.

[4] A. Casimiro, P. Martins, and P. Veríssimo. How to build a Timely Computing Base using Real-Time Linux. In *Proceedings of the IEEE International Workshop on Factory Communication Systems*, pages 127–134, September 2000.

[5] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, November 2002.

[6] T. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *Proceedings of the 15th ACM Symposium on Principles of Distributed Computing*, pages 322–330, May 1996.

[7] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[8] F. Christian and C. Fetzer. The timed asynchronous system model. In *Proceedings of the 28th IEEE International Symposium on Fault-Tolerant Computing*, pages 140–149, 1998.

[9] M. Correia, N. F. Neves, L. C. Lung, and P. Veríssimo. Low complexity Byzantine-resilient consensus. *Distributed Computing*, 17(3):237–249, 2005.

[10] M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In *Proceedings of the Fourth European Dependable Computing Conference*, pages 234–252, October 2002.

[11] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. A realistic look at failure detectors. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 213–222, Washington, USA, June 2002.

[12] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.

[13] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[14] Christof Fetzer. Perfect failure detection in timed asynchronous systems. *IEEE Trans. Comput.*, 52(2):99–112, 2003.

[15] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[16] R. Friedman, A. Moustefaoui, S. Rajsbaum, and M. Raynal. Error correcting codes: A future direction to solve distributed agreement problems? In *International Workshop on Future Directions of Distributed Computing, FuDiCo*, June 2002.

[17] Roy Friedman, Achour Mostéfaoui, and Michel Raynal. Building and using quorums despite any number of process crashes. In *5th European Dependable Computing Conference (EDCC'05)*, Budapest, Hungary.

[18] J.M. Helary, M. Hurfin, A. Mostefaoui, M. Raynal, and Tronel F. Computing global functions on asynchronous distributed systems with perfect failure detectors. *IEEE Transactions on Parallel and Distributed Systems*, 11(9), September 2000.

[19] I. Keidar and S. Rajsbaum. On the cost of fault-tolerant consensus when there are no faults - a tutorial. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 32(2):45–63, 2001. Preliminary version, MIT Technical Report MIT-LCS-TR-821, May 24, 2001.

[20] F. Meyer and D. Pradhan. Consensus with dual failure modes. In *Proceedings of the 17th IEEE International Symposium on Fault-Tolerant Computing*, pages 214–222, July 1987.

[21] A. Mostéfaoui, E. Mourgaya, and M. Raynal. Asynchronous implementation of failure detectors. In *Int. IEEE/IFIP Conference on Dependable Systems and Networks (DSN'03)*, San Francisco (USA).

[22] N. F. Neves, M. Correia, and P. Veríssimo. Solving vector consensus with a wormhole. *IEEE Transactions on Parallel and Distributed Systems*, 16(12):1120–1131, December 2005.

[23] D. Powell. Fault assumptions and assumption coverage. In *Proceedings of the 22nd IEEE International Symposium of Fault-Tolerant Computing*, July 1992.

[24] D. Powell, D. Seaton, G. Bonn, P. Veríssimo, and F. Waeselynk. The Delta-4 approach to dependability in open distributed computing systems. In *Proceedings of the 18th IEEE International Symposium on Fault-Tolerant Computing*, June 1988.

[25] M. Raynal. Short introduction to failure detectors for asynchronous distributed systems. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 36(1):53–70, 2005.

[26] Nicola Santoro and Peter Widmayer. Majority and unanimity in synchronous networks with ubiquitous dynamic faults. In *SIROCCO*, pages 262–276, 2005.

[27] P. Sousa, N. F. Neves, and P. Verissimo. How resilient are distributed f fault/intrusion-tolerant systems? In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, June 2005.

[28] Jan van Leeuwen and Jir Wiedermann. Beyond the turing limit: Evolving interactive systems. In Leszek Pacholski and Peter Ruzicka, editors, *SOFSEM: Theory and Practice of Informatics, 28th Conference on Current Trends in Theory and Practice of Informatics*, volume 2234 of *Lecture Notes in Computer Science*, pages 90–109, Piestany, Slovak Republic, 2001. Springer.

[29] P. Veríssimo. Uncertainty and predictability: Can they be reconciled? In *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 108–113. Springer-Verlag, 2003.

[30] P. Veríssimo and A. Casimiro. The Timely Computing Base model and architecture. *IEEE Transactions on Computers*, 51(8):916–930, August 2002. Supersedes Tech. Rep. DI/FCUL TR-99-2, Dpt. of Informatics, University of Lisboa, May 1999.

[31] P. Veríssimo, A. Casimiro, and C. Fetzer. The Timely Computing Base: Timely actions in the presence of uncertain timeliness. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 533–542, June 2000.

[32] P. Veríssimo and L. Rodrigues. *Distributed Systems for System Architects*. Kluwer Academic Publishers, 2001.

[33] P. Veríssimo, L. Rodrigues, and A. Casimiro. Cesiumspray: a precise and accurate global clock service for large-scale systems. *Journal of Real-Time Systems*, 12(3):243–294, May 1997.

[34] C. Walter, N. Suri, and M. Hugue. Continual on-line diagnosis of hybrid faults. In *Proceedings of the 4th IFIP International Working Conference on Dependable Computing for Critical Applications*, 1994.

[35] L. Zhou, F. Schneider, and R. van Renesse. COCA: A secure distributed on-line certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368, November 2002.