A Distributed Cloud Robotics Middleware Architecture for Trust-minimized Storage and Computation

Alessandro Simovic, Ralf Kaestner and Martin Rufli

***Abstract*— We introduce a novel, distributed architecture utilizing consensual, blockchain-secured computation and verification that enables a scalable, transparent, and semantically interoperable cloud robotics middleware, capable of powering an emerging internet of robots in an automated way, without human intervention.**

Our architecture combines a set of modular key technologies that distinguishes it from existing, centralized robotics middleware approaches whilst enhancing their value. This includes technologies to (i) render data available across a distributed network; (ii) enact re-usability of data and applications, and their interoperability; (iii) universally execute data-manipulating applications yielding reproducible results, independently of the underlying infrastructure; and (iv) enable trust-minimized verification of derived data and the computations generating it. On an implementation level, the proposed middleware architecture integrates with ontologies, content-addressable distributed storage, a blockchain database, and software containerization.

In this paper, we motivate and justify the need for such a system and all underlying building blocks. Design and interplay of the components are explained in detail and embody the main contribution of our work. The benefits of our approach are highlighted by means of a sample application.

I. Introduction

The cumulative number of cognitive connected devices is expected to surpass 100 billion units by the year of 2025, resulting in a trillion-dollar market [1] for the Internet of Things (IoT) and robotics. This rapidly evolving ecosystem is being driven by a truly remarkable advancement in both the technological maturity and commercial deployment of connected platform applications and services over recent years. Mobile robotics has traditionally been characterized by placing an emphasis on on-device cognition supporting independent operation. IoT on the other hand relies on minimal processing power and actuation capabilities on the devices themselves, while outsourcing much of the cognition to external infrastructure. Looking forward, further platform miniaturization and pervasive availability of both, wireless connectivity and cloud as a service offerings engender a new ecosystem. The convergence of mobile robotics and cognitive IoT into the emerging field of Cloud Robotics [2] is therefore the next logical step in an ongoing evolution.

The characteristics of and the scale at which this convergence is happening results in an unprecedented potential for network effects. Ubiquitous availability of big data, paired with recent breakthroughs in data-driven learning approaches may result in a knowledge revolution for cognitive robotic devices; similar in importance to the impact services like Wikipedia and OpenStreetMap had on humankind.

A. Simovic is with the University of Zurich, Switzerland. e-mail: simovic@ifi.uzh.ch

R. Kaestner and M. Rufli are with IBM Research – Zurich, Swizerland. e-mail: {alf, mru}@zurich.ibm.com

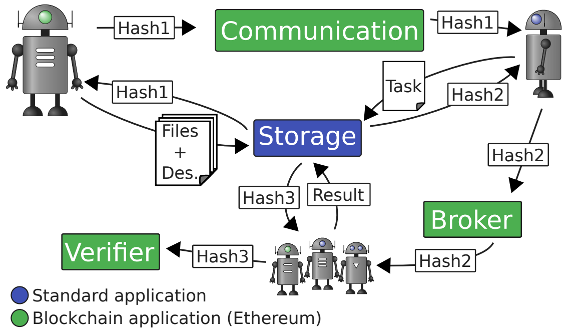

Fig. 1. Overview of the presented cloud robotics middleware architecture, its key components and their interplay.

With the exponentially growing number and interconnectivity of robotic devices however, scalability of the underlying systems becomes a key difficulty. This particularly concerns the interaction and cooperation among heterogeneous sets of robots; and indeed the very sharing of data and knowledge. The issue is that in such a setting, knowledge may no longer be effectively and verifiably curated by existing, efficient central entities alone. Yet no automated, robust and decentralized alternatives are available today. To overcome these challenges and fully benefit from the promise of Distributed Cloud Robotics, the following four key aspects must be addressed:

- **Broad availability and integrity of data and knowledge:** Domain knowledge (consisting of axioms, derived knowledge, program code, and ontologies) and data must be openly available and transparently verifiable.
- **Trustability in data and computation:** Availability of information on its own is not sufficient. Robots need to be able to trust, that computations performed by their peers have been completed correctly. Only then information can form a reliable source of input for further derived operations.
- **Re-usability of and interoperability across applications:** Programs must be modular and semantically interoperable to facilitate re-use across standards.
- **Universal executability of software programs:** Deterministic programs must be universally executable regardless of the underlying hardware, while producing repeatable results. Otherwise outsourcing computation and sharing the respective results is impossible.

A solution encompassing these four aspects would accelerate the development of self-organized networks consisting of heterogeneous platforms; away from the singular setups or relatively small groups of homogeneous platforms collaborating on a specific problem in operation. It would facilitate the sharing of distributed semantically organized information as (partially) open-domain collaborative memory. And ultimately it would support the separation of physical and functional properties within cognitive systems – towards efficiently used and fully decentralized infrastructure, platform and software as a service offerings.

We have devised and implemented a middleware architecture addressing each of these aspects. It constitutes a set of interlocking tools, which provide individual applications interacting with each other with the means for creating and maintaining verifiable computational results in distributed networks. It consists of interfaces to building blocks originating from advancements and recent breakthroughs in diverse fields of science. These include:

- Content-addressable, and thus automatically verifiable, distributed storage that operates with a Distributed Hash Table (DHT),
- A merkle DAG / blockchain enabling proof of record for all data additions and manipulations,
- A base ontology providing the taxonomies and vocabulary necessary for semantic description or encoding of data as well as seamless interoperability of generic software components,
- Software virtualization allowing programs to be packaged, distributed to and executed on various hardware platforms.

Basic interaction of these modules is illustrated in Figure 1. The introduction, description and potential of our approach pose the main purpose of this paper, which we believe to be the first of its kind.

A superficial, but illustrative example application built with our middleware is the enablement of a fully decentralized, transparent and verifiable knowledge base for robots. The application can be thought of as an extension to Roboearth [3] with the following benefits: knowledge generation, storage and manipulation can be trusted and reused among robots even when they themselves do not trust each other – a common scenario in a world wide web for robots. In this setting the entirety of sensor data, derived knowledge and software components is stored in a decentralized manner. All data is cached and shared on a request basis among the robots and other devices participating in the network. Containerized software programs are geared at the manipulation and enrichment of knowledge. The execution thereof may be routed through the system's blockchain interface, making their output visible to and transparently verifiable by others with no further computations necessary. While this just represents one sample application, it should elucidate the motivation behind our approach.

The remainder of this paper is organized as follows: In Section II we review related work, both at the system level and at that of its constituent parts. Section III introduces and provides an overview of the proposed middleware architecture. In Section IV, an illustrative example application operating on the presented architecture is motivated and sketched. Section V summarizes and concludes the paper.

II. Related Work

In this section we review work related to the overall concept of the proposed framework as well as the individual functional parts it is composed of.

A. Frameworks and Middleware Approaches

Both the sharing of knowledge and data in a multi-agent system as well as the outsourcing of computations to the cloud have recently been subject of a growing research effort. One of the first cloud computing frameworks for robotics was DAvinCi [4] with the primary aim of moving the popular Robot Operating System (ROS), and thus the outsourcing of computation, to scalable cloud infrastructure. RoboEarth [5] strives to implement an internet for robots by placing its focus on a shared and high-level knowledge repository. Moreover all information is agnostic to the specific hardware configurations of individual robots interacting with RoboEarth. Rapyuta [3], [6], an advancement of RoboEarth, focuses on universal packaging and remote execution of software via Linux containers.

However, while these approaches enable the generation and sharing of knowledge among robots, as well as the outsourcing of heavy computation to the cloud, they do not promote a fully decentralized knowledge management and neither provide any means to verify outsourced computation to create trust in data outputs among potentially unknown robots. While this is an acceptable solution for an intranet environment, it becomes completely inadequate at the scale and automation level required to power an internet of robots comprised of millions of platforms. Adding to the problem, most of these are likely unknown and potentially distrustful. Our middleware, by contrast, integrates recent advances in distributed storage, communication and universal computing to enable the latter scenario. The remainder of this section covers literature in relation to these aspects.

B. Consensus Algorithms

In distributed software systems, the collective often needs to reach consensus in order to operate properly. Replicated state machines are a common form of conensus algorithms. A copy of the machine's current state is replicated on all nodes, allowing the system to continue operation even in case of node failures. Protocols like ZooKeeper [7], Paxos [8] and Raft [9] use randomly-elected leaders to dictate progress in the system. They are robust as a failing leader node is quickly replaced and the system can continue to operate. Malicious or misconfigured nodes however pose a threat to any leader-based system. This issue is answered by Bitcoin [10]. It is a purely decentralized, transaction-based state machine that is also cryptographically secure. While Bitcoin is centered around its identically named currency, other projects are trying to generalize its concept. One is Ethereum [11], which adds Turing-complete programming to the mix. It allows developers to create Decentralized Applications (Dapps) and to use the blockchain as a storage and message medium. Executing Dapps is thus highly reliable. Fraud as well as third-party inferences are nearly impossible. The transaction costs in Ethereum are relative to the induced computational effort. This limits Dapps to lightweight use-cases.

The combination of distributed storage and lightweight distributed compute coupled with computation outsourcing forms the basis of our middleware architecture.

Replicated state machines are often used as part of distributed storage systems. Hadoop File Storage [12] and Google File System [13] are two of them. However, their design restriction to handle crash failures only limits their use to private data centers. Another distributed cloud storage system is Tahoe-LAFS [14], which can be used even when untrusted nodes are part of the network; a key difference compared to HDFS and GFS. Motivated by Bitcoin's success, two storage solutions that use the blockchain as a conensus-algorithm have been created: Filecoin [15] and Storj [16]. Both are operated by anonymous and untrusted peers that provide storage for the collective. Moreover they address files by a hash of the content rather than their location; a concept commonly referred to as Content-addressable storage (CAS). Downloads from untrusted sources can thus be verified easily because corruptions or modifications alter the file-hashes and -addresses. Finally, there is IPFS [17], a protocol for decentralized and content-addressed file sharing. It is heavily inspired by established P2P applications like BitTorrent [18] and Freenet [19]. For instance, files are exchanged on a tit-for-tat basis. One way of categorizing distributed storage systems is by their incentive model. HDFS, GFS and Tahoe-LAFS are not providing an incentive for correct participation at all. In IPFS there is currently only a local incentive. Its tit-for-tat file exchange strategy motivates individuals to help the peers that can offer something in exchange. Filecoin and Storj go a different route with a global incentive. A public ledger stored on a blockchain forms their consensus-mechanism. „Donating" storage space is rewarded while the allocation thereof is tied to fees.

C. Distributed Computing under Minimum Trust

Similar to distributed storage, the outsourcing of computation to untrusted hardware has a rich history. In the basic case results are only relevant for the entity orchestrating the outsourcing. Well-known Grid Computing applications fall into this category, including SETI@Home [20], Boinc [21] and Folding@Home [22]. In these implementations verification typically remains handled by replicating a single computation on multiple, untrusted peers. The individual outputs are then compared and the correct result determined by majority count. This is despite recent progress that would allow for improved resilience agains Byzantine Faults [23] and increased efficiency via probabilistic proof systems [24], [25]. Today these systems still pose a tradeoff between performance and expressiveness of code and are thus not yet practical for generic real-world use [26], [27].

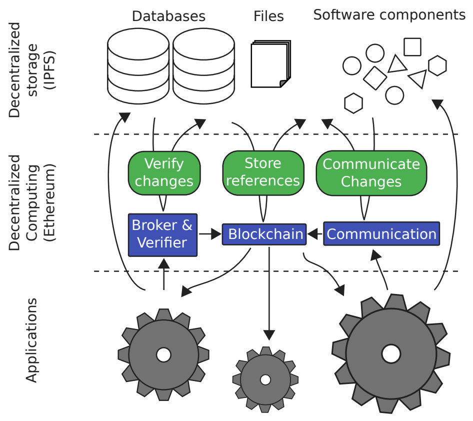

Fig. 2. Overview of the proposed middleware architecture. Robotic devices host and operate a share of the decentralized storage, communication and computing infrastructure. They interact with the network by adding and advertising new datasets, or manipulating existing data to gain further insights that they may elect to share with the collective again.

D. Virtualization and Containerization

The outsourcing of computation requires a a secure and reliable environment on the hosts to operate on. Virtualization on one hand supports security through isolation [28], and this is where hypervisors like Xen [29] and OpenVZ [30] are employed. Both run on bare metal machines and virtualize their resources into so-called virtual machines. Virtualization on the other hand enables portability of applications, a use case that is currently omnipresent in cloud platforms. LXD [31] and Docker [32] are both container hypervisors, where applications, their dependencies and an OS are packed together.

III. Middleware Architecture

The aim of this section is to provide an in depth introduction to the proposed middleware architecture; at the system level, at that of its underlying constituent modules, and at the level of applications building on top of it.

A. Architecture Overview

In a nutshell, our architecture composes a set of interlocking technologies enabling other applications to create, maintain and share an ever increasing amount of verifiable knowledge in a decentralized network. The middleware provides means for an exponentially growing number of robots to interact with each other, share data and knowledge, and to build on top of information generated by others without explicitly knowing or even trusting them individually.

Figure 2 provides an overview and permits various perspectives at the architecture, its modules and their interplay. One such perspective is centered around hardware, i.e. infrastructure. Under the evolving X as a service (XaaS) paradigm, cognitive physical devices can be thought of as a minimal assembly of certain physical modules: physical storage, compute, networking, sensing and actuation, all of which form an infrastructure to complete a range of tasks. As such mobile robots can be thought of as devices composed of all aforementioned hardware parts. Effectively IoT devices are robots with limited compute and typically lack actuation capability. Cloud servers are powerful storage, networking and compute systems. Robots, IoT devices and servers may each host a share of the decentralized software modules that make up the overall system. In addition they can also run cognitive client applications, proprietary or again publicly shared. These applications interact with the framework as illustrated in the bottom third of Figure 2.

Another perspective at our architecture is platform-centric. It permits a view at the various software modules constituting the middleware. Functionally, these consist of decentralized storage and decentralized computing functionality. The proposed architecture comes with interfaces to specific third-party contributions fulfilling these functions. They exist and operate collectively among participating hardware hosts, thereby making up the decentralized network. Its functional modules and their implementations are further detailed in Section III-B.

A third perspective at our approach is from the client applications' point of view, that is, software operating on the physical robotic devices. The aggregate of these applications defines the capabilities, or in other words the collective intelligence of connected platforms. As such, a robot may consume or generate data via the middleware and inferred insights can again be shared with the collective. Details on data representation and manipulation within the framework are treated in Section III-C, and Section III-D respectively.

B. Modules and Interfaces

Our middleware is composed of a set of tools which provide client applications with the means for creating, maintaining and sharing verifiable computational results. To this end it provides interfaces to appropriate storage, communication and compute modules. Each of them has been selected as a critical element in supporting the framework's design goals: Decentralized content-addressable storage serves as a decentralized repository holding all shared information. A decentralized timestamping server enables robots to persistently advertise new data. Finally, lightweight decentralized computing functionality in combination with deterministic and platform-agnostic heavyweight application executability allow for the outsourcing of resource-intensive data processing tasks whose results remain verifiable and can thus be trusted.

1) Decentralized Content-addressable Storage: The middleware's decentralized storage is expected to hold the entirety of accumulated collective data and knowledge, while operating at a bare-bone object store level. This ranges from raw, uninterpreted sensor data via model representations to semantically encoded knowledge and even software modules. Robots need to be able to store and/or retrieve arbitrary information to and from the storage, the interpretation of which is relegated to higher level functionality (such as databases and indexes). The framework interfaces to InterPlanetary File System (IPFS), a Content-addressable storage (CAS), as the framework's storage module. In CAS systems files are addressed by their content rather than location; file addition

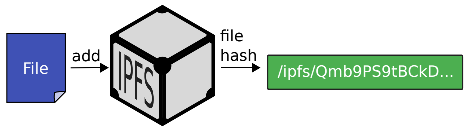

Fig. 3. Files added to IPFS are addressed and retrievable by their SHA256 hash. IPFS maintains a distributed hash table listing peers claiming to hold certain files, on top of which it operates a file exchange mechanism.

is illustrated in Figure 3. Typically, a content-based hash is generated for every file and is used as its address.

With hash-based file retrieval downloads from untrusted sources are trivially verifiable by calculating the retrieved file's hash and comparing it to the one used for the request. Furthermore it is impossible to unnoticeably modify or corrupt files. Every modification automatically results in a new hash-address. Put differently, if the hash of a file is known, it is guaranteed that the retrieved file is exactly the one that has been requested. Hash-based lookup makes CAS well suited for long-term storage, since every file is addressed uniquely and independently of its current location. IPFS maintains a distributed hash table of the files stored, on top of which it operates a file exchange currently based on a tit-for-tat strategy. This makes caching popular files attractive for clients. Data availability is thus directly linked to its popularity. By implication, files without value to the network are forgotten after some time. In that sense the combined storage space of all robots represents an interest-based cache.

2) Trust-minimized Decentralized Communication and Timestamping: The downside of CAS is that storing data does not make it readily visible to and discoverable by others. Before a robot can access a file, its existence needs to be made known to it. Since all data is serialized into files of suitable formats (e.g., RDFXML for semantic knowledge) and stored in IPFS, it suffices to communicate only the hash-address of a file to a connected platform.

While many forms of communication among peers in a decentralized network exist, the design criteria of our cloud robotics system encourage a broadcast-based communication. The use of a blockchain database in this setting results in messages being tamper-proof and cryptographically signed. And since the information in blockchains is by design ordered chronologically, it is possible to find out who advertised specific data first. This opens up possibilities for various high-level applications built on top of our approach, such as file ownership. These advantages are partially offset by limited bandwidth and the potential incurrence of transaction fees. By only storing the content-hash of advertised data, bloat on the blockchain can be mitigated, however. For practical reasons our present implementation of the middleware employs an Ethereum Dapp for communication as illustrated in Figure 4. At the same time, Ethereum's blockchain forms an integral part of the result verification pipeline; more details are provided in Section III-D.

3) Trust-minimized Distributed Computing: As described in Section II, decentralized verifiability of computational results makes a blockchain-based computer necessary. For

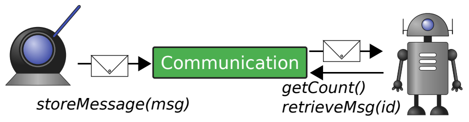

Fig. 4. Communication between platforms is performed through an Ethereum blockchain application. Any connected robot is able to store new messages there, which are then visible to the entire distributed network.

this purpose our framework interfaces with Ethereum [11] via a set of distributed applications. Ethereum offers a decentralized virtual machine, which uses a blockchain as its underyling storage and message platform. Distributed applications, so called Dapps, running in Ethereum's virtual machine are programmed using blockchain transactions. Due to performance limitations inherent to that design, heavy computations – including data manipulations – need to be outsourced. Only their output is verified, for example via replicated executions, on the blockchain.

4) Platform-agnostic Application Execution: Heavy-weight code execution thus needs to be performed off-chain on device hardware. The approach for verifying outsourced computations as explained in Section III-D is an integral part of the framework's trust model and requires the software modules to be universally executable whilst yielding reproducible output. Platform architecture as well as different operating system configurations make universal computation a challenging objective, however. The concept of containerization addresses this issue. Within our approach we therefore distribute software modules in the form of runtime containers with Docker [32]. Since these containers include their own bare-bone operating system, they can be deployed and executed universally, independent of the host system's configuration. Analogous to other forms of data and knowledge, these containers may be stored in the distributed object store described in Section III-B.1, and any robot in the network can obtain and execute them.

C. Semantic Data Representation

Our middleware establishes a view on two fundamental categories of data: Axiomatic knowledge and verified data / knowledge, which in turn includes sensor readings, programmatic source code, and various forms of data derived thereof.

1) Axiomatic Knowledge: Axiomatic or postulated knowledge refers to semantic statements that cannot be proven from within the system itself (i.e. the virtual or the physical world), but are nonetheless self-evident or commonly accepted. This can be common knowledge or prior information that is treated as a given fact. Every subsequent use of axioms within the middleware operations then generates derived data, which can be verified. In that sense axiomatic knowledge will always form the first element in a chain of verifiable computations.

Within our middleware approach, ontologies form the main source of axiomatic knowledge. They are typically created by humans and hold axioms specific to a domain. For our implementation of the concept, we developed a

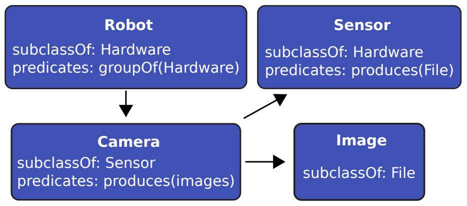

Fig. 5. Excerpt of the base ontology employed in our approach to describe files. Here, an image file is linked to the camera sensor and the robot that produced it.

basic but extendible Web Ontology Language (OWL) [33] ontology, an excerpt of which is illustrated in Figure 5. Additional ontologies can extend the semantic capabilities of applications. The presented approach does not restrict the use of other ontologies such as Semantic Robot Description Language (SRDL) or the ones provided by KnowRob [34], [35]. They can be used in conjunction with its ontology. The middleware stores axiomatic knowledge and makes it available via its distributed storage interface. To this end axiomatic statements are serialized into an RDFXML file stored in IPFS like any other file.

Axiomatic knowledge is used to semantically describe sensor data, software applications, and the computational results as introduced above. For generic data, which is serialized and stored in files, the corresponding semantic information includes attributes such as the file format and size, type of its content and, most importantly, the file's URI. The framework's ontology is also used to describe the software applications for data processing introduced in Section III-B.4. Applications are characterized by stating the kind of input they require as well as the output they produce. A description of an application for resizing images for example could read: *Input and output of the application are images that are encoded in the JPEG format and have attributes width and height set.*

Since an ontology provides abstraction of information, our approach also uses it to configure software modules. Returning to the example of image resizing, a configuration would at least have to include the output type “image” as well as the file format and the desired image attributes “width” and “height” or equivalent concepts. The application is then configured by passing it this description. Our approach for configuring software permits robots that do not know implementation specifics of applications to nonetheless utilize them in a generalizable way. This is made possible because all information relevant to the application is located in the corresponding high-level semantic descriptions.

2) Verified Data / Knowledge: Verified data and knowledge includes sensor readings, algorithmic recipes, and data generated thereof through computation.

Robotic sensing devices produce streams of information, typically in the form of time series. As an example, let us consider a time series built from sequential GPS measurements. This data is captured in a raw, uninterpreted form. Among various other attributes its description can include the sensor classification and a unique identification number,

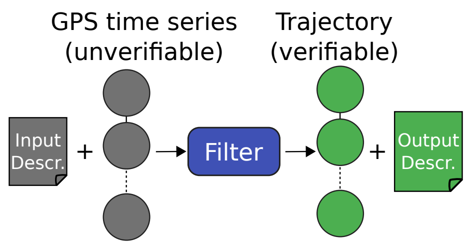

Fig. 6. Example application where a sensor dataset is filtered to produce a smoothed trajectory. Both input and derived output datasets are described semantically for further access and stored within the distributed storage. The filter is essentially a program, which itself is a derivation of its corresponding source code.

1. Knowledge manipulation task is sent to verifier (including participation rules and reward payout schedule)
2. Agents register with verifier as prover candidates and stake a security deposit
3. Provers are chosen randomly by verifier
4. Provers perform computations yielding a result *R*
5. Provers chose a secret *nonce* to obfuscate result *b = h(R,nonce)*
6. Provers submit their respective blinded result *b* to the verifier
7. Provers reveal their nonce to verifier together with the result *R*
8. Verifier validates all blinded results from step 6
9. Verifier elects final result (either directly, or by forming a dependent knowledge manipulation task from the submitted results)

Fig. 7. Algorithm describing interaction between provers and the system's distributed verifier to perform computation outsourcing in a cloud of robots.

the file format and the robot it was attached to at the time of recording. Much of the information in the object store will originally be inserted as sensor data; the other possibility being axiomatic knowledge as described above. To verify sensor data, the collective may produce evidence of its integrity, e.g., by deriving higher-level models in the form of maps, which can then be verified directly. This approach is illustrated in Figure 6. Due to the demand in resources, it would commonly involve off-chain computations whose results must again be verified on-chain. This case will further be discussed by the sample application presented in Section IV.

Algorithmic recipes may be present in the form of application source code, be it compilable (e.g., C++) or directly interpretable (e.g., Python). Under the open source policy, it may directly be verified by human experts. Compiled source code and application containers are a form of derived knowledge. Compilation and containerization may in turn pose an off-chain effort whose results must be verified on-chain, e.g., by replication and comparison of container content hashes. All objects and containers generated from algorithmic recipes include a pointer to the respective source code in their descriptions.

D. *Trust-minimized Distributed Computing*

Data processing forms the central element of any robotic application interacting with and through our middleware. By data processing we refer to computations transforming input data, either in the form of raw sensor data or as derived data according to semantically encoded instructions. The computations then produce derived data, together with the corresponding semantic descriptions. Figure 1 summarizes the corresponding data and information flows: Robots may add new files and their semantic descriptions to the decentralized storage as depicted in the center of the figure. New datasets or novel derived information are advertised by communicating the hash and thus the address of the description file to the network. Other platforms may download the description and explore its contents. They may further come to the conclusion that some of it is useful to advance their goals. The robots may then create a manipulation task, which they either process privately for their own consumption, or have it processed publicly by the collective network.

The common scenario for verifying outsourced computations features provers and a verifier. The verifier tasks the provers with a computation and later verifies it. In a distributed setting the situation is complicated by the fact that it is not sufficient for only the original verifier to know about a computation's correctness. Rather every single robot in the network needs to be convinced of its correctness without executing the computation again. As discussed in Section II verification in computation outsourcing often involves replicated work.

When offloading computations to other platforms, a robot needs to create a work description and encode it semantically. In the description, the exact application container needs to be referenced together with its configuration, which includes the input files to be used. On a side note, compiling and assembling the software containers themselves could be another outsourced computational effort as mentioned in Section III-C.2. In Section III-D we propose to use Docker containers for packaging application software, ensuring repeatable executability irrespective of the underlying system. Containers and data are referenced with their respective Unique Resource Identifiers (URIs) pointing to files in the decentralized storage. The robot then tasks a dedicated Ethereum Dapp – in Figure 1 denoted as "Broker" – to advertise the computational task to interested provers with spare resources. This may include a monetary reward, paid out to provers that submitted the majority result. As such it is possible for an economic market for outsourced computations to emerge.

The Dapp in turn manages the provers and the verifier. Prover candidates register with the Dapp. They confirm their intent by depositing a monetary amount called *stake* in accordance with the computational task's requirements. This strategy has recently been proposed by the blockchain community under the notion of proof-of-stake [36].

Upon successful execution, application containers provisioned by provers single-handedly create semantic descriptions of their output. Next, provers submit the description to the Dapp for verification in a two-part procedure, which is explained in Figure 7. If the majority of the results supplied by the provers are found to be in agreement by the Dapp, it will be accepted and the associated reward paid out. Otherwise, the staked security deposits are withheld.

Depending on the nature or implementation of a computational task, results may be either deterministic or probabilistic. Deterministic results usually occur as outcome of purely deterministic algorithms, i.e., those algorithms which do not employ random sampling, and can be compared efficiently and directly within the cost-limited scope of a Dapp. Note that the determinism assumption would also hold for results generated from heterogeneous devices with differing numerical precision: These typical precisions are part of the collective knowledge and can be used to define epsilon bounds for comparison.

In many robotic problems, however, tractability or computational effort dictates the involvement of random sampling in an algorithmic approach. In cases where the results submitted by provers are known to be non-deterministic, the verifier may chose to interpret them as samples of a probability distribution whose parameters are (partially) unknown. Consequently, a dependent off-chain knowledge manipulation task can be created with the goal to estimate the unknown model parameters and to subsequently classify between accepted and unaccepted results. The classified inputs then pose a deterministic result which would again be validated on-chain. The stakes of the provers and the rewards must be withheld until the delayed verification came to a deterministic conclusion.

Our approach does not explicitly enforce the number of replications and the thereby gained confidence in the results. Robots that outsource a computation are free to specify this parameter. Having said this, other robots can view the number of replicated computations on the blockchain. They can thus determine whether the deduced probability of the derived result being correct suffices their own needs.

In order to mitigate the risk of having malicious robots collaborate to falsify results, the Dapp randomly selects interested provers from a sufficiently large set. Having the decentralized verifier perform random selection is a tricky process because Ethereum Dapps are deterministic – though it is possible to create a Pseudo random number generator (PRNG). In our Dapp implementation we employ the hash of a predetermined upcoming block as the seed of the PRNG. Using a future block is necessary to prevent abuse of the system. Otherwise the outcome of the PRNG could be gamed opportunistically.

As storing data on the blockchain is costly, the full result of a computation is never transmitted. Instead only a hash of the result is submitted to the verifier. To be more precise, verification uses the hash-based URI of the semantic descriptions of the result, which is generated by the application. In Figure 1 this URI is denoted by "Hash3". Despite serving the verification process, the hash is also everything that robots later on require to obtain the results from the storage. This is owed to the fact that result descriptions also include complete indexes of all newly generated files and their respective URIs and descriptions, despite other logical information. Verifying results by only comparing their hashes bears one more advantage: Computations can be verified without obtaining the resulting files themselves.

IV. Application Outline

In this section the sample application of a distributed, transparent, and verifiable knowledge base for robots introduced in Section I is outlined in more detail. Information regarding possible implementations and examples for flow of information have been incorporated in the outline.

The presented middleware architecture can power a complete cloud platform for robots. Typically such platforms are used to gather sensor data and semantic information, aiding centralized task control. Two existing cloud engines for robots are RoboEarth [5] and Rapyuta [3], [6]. With these engines, knowledge and data can be abstracted and exchanged between individual robots. Furthermore these cloud engines allow their peers to outsource computations, therefore reducing the hardware requirements for mobile robots. In the following we sketch an approach with capabilities similar to RoboEarth, but impose fewer restrictions regarding its implementation and operation.

All knowledge and data described in Section III-C is added to the middleware's decentralized storage. Advertising the data makes it known by and therefore accessible to every platform. Also, since application-specific software is stored like any other data, robots can download and execute them unrestrictedly. Applications can either be run locally for personal needs only or outsourced to the cloud's network when the results need to be globally verifiable. It is up to every robot to decide whether or not verification is required. Moreover, each robot is free to consume existing information passively, i.e. to not share any newly generated knowledge or application with the collective.

Building cloud applications on top of our approach implies various benefits:

- Cloud applications can be fully decentralized, making server hardware optional.
- Hardware rented from CSPs can be involved but does not need to be trusted because of the middleware's support for verifiable computations.
- Distinguishing robots and non-robotic devices, e.g., server hardware, is no longer necessary. Both roles can be taken on selectively by everyone.
- Existing cloud engines such as RoboEarth and Rapyuta can be operated in an open public mode where anonymous robots are free to contribute.
- By performing computations for the network, idle hardware can be optimally used.
- Bandwidth consumption is reduced in scenarios where interconnected platforms require the same data. The possibility to obtain data from any robot diminishes bottlenecks posed by slow communication infrastructure in mobile applications.

There are however some drawbacks associated with our approach that should not remain unmentioned: Central task control with the proposed middleware architecture is not possible by default. A subset of the collective can however decide to trust a particular peer for taking on the controller's duties. This would then be part of a top-level application, which could be realized with the tools provided by our middleware. Outsourcing tasks that do not need verification might further be found to not be cost-beneficial with our approach, the reason being that in our current system computations are verified by replication. The accumulated costs of an outsourced execution will therefore multiply as compared to a solution in which computation is performed only once on privately provisioned hardware.

V. Conclusion and Future Work

In this work, we presented a novel approach to a scalable and semantically interoperable cloud robotics middleware for trust-minimized infrastructure. We have demonstrated that, with existing technologies, each of its components can be designed trustworthily under the key aspects of decentralization and transparency. Semantic formalization of all data and applications maximize their utility, improve interoperability, and provide elegant means of verifying the outcome of computations. The framework's hardware-agnostic architecture further enhances interoperability and drastically increases scalability. The presented on-chain verification approach allows the correctness of a result to be determined without actually acquiring it. For mobile robots with limited storage and bandwidth, this is immensely beneficial.

Through state-of-the-art decentralized consensus, we substantiate the potential of our architecture to lay the foundations of a truly open cloud robotics knowledge base without sacrificing credibility of its content. Current implementations restricted to private environments can thus be scaled to public infrastructure and generate larger visibility and impact.

References

[1] J. Manyika, M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin, and D. Aharon. (2015, Jun.) Unlocking the potential of the internet of things. Available online. [Online]. Available: http://www.mckinsey.com/business-functions/business-technology/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world

[2] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *Automation Science and Engineering, IEEE Transactions on*, vol. 12, no. 2, pp. 398–409, Apr 2015.

[3] D. Hunziker, G. Mohanarajah, M. Waibel, and R. D'Andrea, "Rapyuta: The RoboEarth cloud engine," in *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013, pp. 438–444.

[4] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit, "Davinci: A cloud computing framework for service robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2010, pp. 3084–3089.

[5] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "Roboearth," *IEEE Robotics and Automation Magazine*, vol. 18, no. 2, pp. 69–82, Jun. 2011.

[6] G. Mohanarajah, D. Hunziker, M. Waibel, and R. D'Andrea, "Rapyuta: A cloud robotics platform," *IEEE Transactions on Automation Science and Engineering*, February 2014.

[7] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems." in *USENIX Annual Technical Conference*, 2010, pp. 145–158.

[8] L. Lamport, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, Nov. 2001.

[9] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX Annual Technical Conference*, 2014, pp. 305–319.

[10] S. Nakamoto. (2008) Bitcoin: A peer-to-peer electronic cash system. Available online. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[11] (2016, Feb.) Ethereum Whitepaper. Available online. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[12] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[13] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *Operating Systems Review (SIGOPS)*, vol. 37, no. 5, pp. 29–43, Oct. 2003. [Online]. Available: http://doi.acm.org/10.1145/1165389.945450

[14] (2016, Mar.) Tahoe-lafs. https://www.tahoe-lafs.org. [Online]. Available: https://www.tahoe-lafs.org

[15] (2015, Jul.) Filecoin: A cryptocurrency operated file storage network. Available online. [Online]. Available: http://filecoin.io/filecoin.pdf

[16] S. Wilkinson. (2015, Dec.) Storj - a peer-to-peer cloud storage network. Available online. [Online]. Available: https://storj.io/storj.pdf

[17] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, Jul. 2014.

[18] (2016) Bittorrent. http://www.bittorrent.com/. [Online]. Available: http://www.bittorrent.com/

[19] I. Clarke, O. Sandberg, M. Toseland, and V. Verendel. (2010) Private communication through a network of trusted connections: The dark freenet. Available online. [Online]. Available: https://freenetproject.org/about.html#papers

[20] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: An experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002. [Online]. Available: http://doi.acm.org/10.1145/581571.581573

[21] (2016, Jan.) Boinc. http://boinc.berkeley.edu/. [Online]. Available: http://boinc.berkeley.edu/

[22] V. S. Pande. (2016) Folding@home. Available online. [Online]. Available: https://folding.stanford.edu

[23] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002. [Online]. Available: http://doi.acm.org/10.1145/571637.571640

[24] O. Goldreich, *Probabilistic Proof Systems: A Primer*. Now Publishers Inc, Jun. 2008.

[25] M. Sudan, "Probabilistically checkable proofs," *Communications of the ACM*, vol. 52, no. 3, pp. 76–84, Mar. 2009. [Online]. Available: http://doi.acm.org/10.1145/1467247.1467267

[26] M. Walfish and A. J. Blumberg, "Verifying computations without reexecuting them," *Communications of the ACM*, vol. 58, no. 2, pp. 74–84, Jan. 2015. [Online]. Available: http://doi.acm.org/10.1145/2641562

[27] R. Canetti, B. Riva, and G. N. Rothblum, "Practical delegation of computation using multiple servers," in *ACM Conference on Computer and Communications Security (CCS)*. New York, NY, USA: ACM, 2011, pp. 445–454. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046759

[28] S. Mansfield-Devine, "Security through isolation," *Computer Fraud & Security*, vol. 2010, no. 5, pp. 8–11, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S136137231070052X

[29] (2016) Xen hypervisor solution. https://www.xenproject.org. [Online]. Available: https://www.xenproject.org

[30] (2016) Openvz virtuozzo containers wiki. http://openvz.org. [Online]. Available: http://openvz.org

[31] (2016) Linux containers introduction. Available online. [Online]. Available: https://linuxcontainers.org/lxd/

[32] (2016) Docker. https://www.docker.com. [Online]. Available: https://www.docker.com/what-docker

[33] (2012, Dec.) Owl 2 web ontology language document overview. Available online. W3C OWL Working Group. [Online]. Available: https://www.w3.org/TR/owl2-overview/

[34] M. Tenorth and M. Beetz, "Knowrob - knowledge processing for autonomous personal robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2009, pp. 4261–4266.

[35] M. Tenorth, A. C. Perzylo, R. Lafrenz, and M. Beetz, "The roboearth language: Representing and exchanging knowledge about actions, objects, and environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2012, pp. 1284–1289.

[36] A. Poelstra. (2016) On stake and consensus. Available online. [Online]. Available: https://download.wpsoftware.net/bitcoin/pos.pdf