Constructing Distributed Time-Critical Applications Using a Vector Symbolic Architecture

Chris Simpkin*, Ian Taylor*,
Graham Bent†, Geeth de Mel†, Raghu K Ganti‡
*School of Computer Science and Informatics, Cardiff University
†IBM Research UK
‡IBM Research USA

Abstract—**Time-critical analytics applications are increasingly making use of distributed service interfaces (e.g., micro-services) that support the rapid construction of new applications by dynamically linking the services into different workflow configurations. Traditional service-based applications, in fixed networks, are typically constructed and managed centrally and assume stable service endpoints and adequate network connectivity. Constructing and maintaining such applications in dynamic heterogeneous wireless networked environments, where limited bandwidth and transient connectivity are commonplace, presents significant challenges and makes centralized application construction and management impossible. In this paper we present an architecture which is capable of providing an adaptable and resilient method for on-demand decentralized construction and management of complex time-critical applications in such environments. To achieve this goal, services must be discovered and orchestrated in a decentralized way with the minimum communication overhead whilst providing detailed information about the workflow - tasks, dependencies, location, metadata, and so on. The approach uses a Vector Symbolic Architecture (VSA) to compactly represent an application as a single semantic vector that encodes the service interfaces, workflow, and the time-critical constraints required. We show that these vectors can be used to dynamically construct and run applications using services that are distributed across an emulated Mobile Ad-Hoc Wireless Network (MANET). Scalability is demonstrated via an empirical evaluation.**

Index Terms—**Decentralized Workflows, Vector Symbolic Architecture, Machine Learning, Dynamic Wireless Networks**

I. Introduction

Time-critical analytics applications are increasingly making use of distributed service interfaces (e.g., micro-services) that support the rapid construction of new applications by dynamically linking the services into different workflow configurations. In traditional Service Oriented Architectures (SOA), operating over fixed networks, the underlying TCP/IP backbone guarantees sufficiently stable service endpoints and connectivity to facilitate the construction and management of multi-service applications using centralized management schemes. Using such schemes also supports the dominant approach to service discovery and matching, which is based on the use of centralised service registries to provide a catalogue of services, using formal ontologies to facilitate service definition and service matching. Such approaches, while often very effective, incur overhead in terms of knowledge engineering effort required to create them [1]. For time-critical applications, centralized control also allows multiple workflow tasks to be load-balanced, scaled, and optimized across multiple heterogeneous compute resources so that their Quality of Service (QoS) and Quality of Experience (QoE) requirements can be fulfilled.

In decentralized environments, such as Mobile Ad Hoc Wireless Networks (MANETs) [2, 3, 4], constructing and running applications that support group-oriented collaborative applications (e.g., multi-user chats) or distributed analytics [5, 6], introduces a much more diverse set of requirements. In such environments end point stability and connectivity remain limited and transient and it becomes impractical, if not impossible, to support centralised service registries and to manage workflows executing at the edge. Time-critical applications operating in such environments introduce further complexity requiring a capability for applications to rapidly reconfigure themselves in the event of change, so that their QoS/QoE requirements can be satisfied. There is therefore a need for new methods that can enable application construction and workflow orchestration without the need for a central point of control.

Micro-services are a variant of the SOA architectural style that structures an application as a collection of loosely coupled services that can be linked in different workflow configurations. They overcome the complexity of ontology-based service composition by having much simpler interfaces (usually RESTful interfaces) which enables these services to be developed by multiple parties without rigid standardisation of service description templates. However, the lack of a formal template means that a mechanism is required that will allow semantic matching/discovery of the appropriate micro-services required for a given application. This mechanism needs to be *compact*, in order to minimize bandwidth requirements, and *flexible*, in order to minimize the knowledge engineering overhead required to generate the service descriptions. In this paper we describe a method by which these objectives can be achieved using the capabilities of Vector Symbolic Architecture (VSA) representations.

Vector Symbolic Architectures (VSAs) [7, 8, 9, 10] are a family of bio-inspired methods for representing and manipulating concepts and their meanings in a high-dimensional vector space. They are a form of distributed representation that enables large volumes of data to be compressed into a fixed size feature vector in a way that captures associations and similarities as well as enabling semantic relationships between data to be built up. Such vector representations were originally proposed by Hinton [11] who identified that they have recursive binding properties that allow for higher level semantic vector representations to be formulated from, and in the same format as, their lower level semantic vector components. As such they are said to be semantically self-describing. Vector unbinding provides a means of retrieving the sub-feature vectors from which it was built.

VSAs are also capable of supporting a large range of cognitive tasks such as; *(a)* Semantic composition and matching; *(b)* Representing meaning and order; *(c)* Analogical mapping; *(d)* and Logical reasoning. They are highly resilient to noise and they have neurologically plausible analogues which may be exploited in future distributed cognitive architectures. Consequentially they have been used in natural language processing [12, 13, 14] and cognitive modeling [15, 16].

Our hypothesis is that a Vector Symbolic Architecture can be used to define a rich and yet compact encoding that will enable the representation of: service descriptions; decentralized service and workflow discovery; distributed workflow execution. This will enable the ability to perform semantic matchmaking and reasoning on service descriptions and service compositions (i.e., workflows).

This paper is a shortened vesion of [17] in which we extend the approach taken in [18] and show how binary VSAs can be used to achieve the following:

- **Encoding workflows:** We discuss and describe how VSA vectors in the form of role-filler pairs can be used to successfully encode both functional and QoS/QoE components of service descriptions. In addition we show how such service descriptions can be encoded via chunking to encode very large sequences of services.
- **Scaling through recursive binding (chunking):** To address scalability, we extend VSAs using a hierarchical vector binding scheme that is capable of representing multiple levels of semantic abstraction (workflow and sub-workflows/branches) into a single vector. We demonstrate empirically, that this scheme can scale to tens of thousands of vectors while maintaining semantic matching, which is adequate for representing most workflows.
- **Representing workflow primitives:** We extend the encoding scheme to support directed acyclic graph (DAG) workflows having one-to-many, many-to-many, and many-to-one connections.
- **Distributed discovery and orchestration:** We show how our VSA encoding scheme can be used for distributed discovery and orchestration of complex workflows. Workflow vectors are multicast to the network and participating services, extended with a simple cognitive layer that can interpret and exchange the vectors, compute their own compatibility and offer themselves up for participation in the workflow. We show how a delayed response mechanism, based on the degree of semantic match, can be used for selection of the best available micro-service for a particular workflow step based on both functional and QoS/QoE requirements while minimizing the bandwidth required for negotiation and selection of such.

The rest of the paper is structured as follows: Section II presents an introduction to VSA mathematical operations and VSA structures; Section III describes and discuses how VSA can be used to build semantic representations of services and their QoS; Section IV outlines how VSA can encode complex workflows and explains how decentralized workflow execution is performed; Section V describes the test-cases used to evaluate of our VSA workflow architecture; Section VI describes the outcome of the test-case evaluation; Finally, Section VII concludes with a short summary and a discussion of the future directions of this research.

II. Vector Symbolic Architecture, Basic Operations

VSAs use hyper-dimensional vector spaces in which the vectors can be real-valued, such as in Plate's Holographic Reduced Representations (HRR) [7], typically having N dimensions ($512 \leq N < 2048$), or they can be large binary vectors, such as Pentti Kanerva's Binary Spatter Codes (BSC) [9], typically having $N \geq 10,000$. For the work here, we have chosen to use Kanerva's BSC but we note that most of the equations and operations discussed should also be compatible with HRRs [15].

Typically, when using BSC, a basic set of symbols (e.g., an alphabet) are each assigned a fixed, randomly generated hyper-dimensional binary vector. Due to the high dimensionality of the vectors the basic symbol vectors are uncorrelated to each other with a very high probability. Hence, they are said to be *atomic* vector symbols [9]. Vector *superposition* is then used to build new vectors that represent higher level concepts (e.g., words) and these vectors in turn can be used to recursively build still higher level concepts (e.g., sentences, paragraphs, chapters...). These higher level concept vectors can be compared for similarity using a suitable distance measure such as Normalised Hamming Distance (HD).

If two high level concept vectors contain a number of similar sub-features, such vectors are said to be *semantically* similar, for example, we can create compound objects analogous to data structures as follows:

$$\mathbf{P1_v} = \mathbf{John_v} + \mathbf{Charles_v} + \mathbf{55yrs_v} + \mathbf{T2Diabetic_v}$$
$$\mathbf{P2_v} = \mathbf{Lucy_v} + \mathbf{Charles_v} + \mathbf{55yrs_v} + \mathbf{T2Diabetic_v}$$
$$\mathbf{P3_v} = \mathbf{Greg_v} + \mathbf{Charles_v} + \mathbf{34yrs_v} + \mathbf{T2Diabetic_v}$$

where + is defined as the *bitwise majority vote* operator.

HD can be used to compare such vectors *without* unpacking or decoding the sub-features. An issue arises, however, when using superposition to build compound vectors in this way because such compound vectors behave as an unordered *bag of features*. Thus, if we have,

$$\mathbf{P4_v} = \mathbf{Charles_v} + \mathbf{Smith_v} + \mathbf{55yrs_v} + \mathbf{T2Diabetic_v}$$

Then, $P4_v$ would be equally similar to $P1_v$ as is $P2_v$ despite the obvious difference in the record. In order to resolve such issues, VSAs employ a *binding* operator that allows vector values such as $Charles_v$ and $55years_v$ to be associated with a particular *field name*, or *role*, within the data structure.

Bitwise XOR is used for both *binding* and *unbinding* with BSC because it is invertible, commutative and distributive over superposition [9, page 147]. This means that both *roles* and *fillers* can be retrieved from a *role-filler* pair without any loss. Due to the distributive property the same method can be used to test for sub-feature vectors embedded in a compound vector as follows:

$$Z = X \cdot A + Y \cdot B \tag{1}$$

$$X \cdot Z = X \cdot (X \cdot A + Y \cdot B) = X \cdot X \cdot A + X \cdot Y \cdot B \tag{2}$$

$$X \cdot Z = A + X \cdot Y \cdot B \tag{3}$$

Examination of 3 reveals that vector **A** has been exposed, thus, if we perform $HD(X \cdot Z, A)$ we will get a match. The second term $X \cdot Y \cdot B$ is considered noise because $X \cdot Y \cdot B$ is not in our known *vocabulary* of features or symbols.

When a role and value are bound together this is equivalent to performing a mapping or *permutation* of a vector's value elements within the hyper-dimensional space so that the new vector produced is uncorrelated to both the role and filler vectors. For example, if $V = R \cdot A$ and $W = R \cdot B$ then R, A and B will have no similarity to V or W. However, comparing V with W will produce the same match value as comparing A with B. In other words, if A is closely similar to B then V will be closely similar to W because *binding* preserves distance within the hyper-dimensional space [9, page 147].

We can now rephrase our *person* record in order to differentiate sub-features within the record, for example, we can formulate $P1_v$ as:

$$\mathbf{P1_v = FN_r \cdot John_v + SN_r \cdot Charles_v + Age_r \cdot 55years_v + Health_r \cdot T2Diabetic_v}$$

This clearly resolves the incorrect matching between $P1_v$ and $P2_v$ with $P4_v$. To test $P1_v$ for the surname $Charles_v$ we perform,

$$HD(SN_r \cdot\ P1_v,\ Charles_v) \tag{4}$$

III. Building Semantic Vector Representations of Services and QoS

Having shown that we can use VSAs to represent data structures, we now consider how to represent service descriptions and their corresponding QoS as symbolic vectors. Reviewing that X_r represents an *atomic* role vector and Y_v a value vector and that $X_r \cdot Y_v$ is a role-filler pair that binds the category X_r to the filler value Y_v and enables later matching and retrieval of values by specific categories, our current scheme employs the following format:

$$Z_x = Serv_r \cdot Serv_v + Resource_r \cdot ResP_v + QoS_r \cdot QoS_v \tag{5}$$

where

- Z_x is the resultant composite service vector;
- $Serv_r \cdot Serv_v$ is the vector representation of the functional description of the service;
- $Resource_r \cdot ResP_v$ is a vector embedded into a request that points to any needed external resources. This is not part of a service's self-description but allows a matching service to locate any external resources specified by a requester; and
- $QoS_r \cdot QoS_v$ is a vector representing either the requester's QoS requirements or the current QoS value for a specific service.

1) Building the Description Vector: $Serv_v$ is itself comprised of symbolic vectors that semantically describe the essential elements of a service, in terms of role and filler pairs that are needed to find a match. To illustrate how this is achieved we use an example of relatively simple service description comprising service name, inputs, outputs, and a functional description of the service, for example:

$$Serv_v = Inputs_r \cdot Inp_v + Name_r \cdot Name_v + Desc_r \cdot Desc_v\ + Outputs_r \cdot Out_v \tag{6}$$

Where

- $Inputs_r \cdot Inp_v$ describes the required inputs;
- $Name_r \cdot Name_v$ a vector encoding of the service name;
- $Desc_r \cdot Desc_v$ a vector encoding of the service description [1]; and
- $Outputs_r \cdot Out_v$ describes the required outputs.

Again the filler component of these vectors can be comprised of other symbolic vectors. In considering Inp_v, Out_v we want to encode these values so that we get flexible matching. For example, if our service, Z_x, has three float inputs and one bitmap input we might encode this as:

$$Inp_v = One_r \cdot Float_r + Two_r \cdot Float_r + Three_r \cdot Float_r + One_r \cdot BitMap_r \tag{7}$$

One_r, Two_r, $Three_r$ are atomic role vectors representing numbers. This simple scheme seems adequate for representing input and output descriptions because micro-services typically do not have a large number of inputs and outputs. More complex input and output descriptions can be encoded via embedding further role-filler pairs. The above vector is a bag representing the inputs that enables flexible matching. If the input part of a request vector is encoded as:

$$InpReq_v = One_r \cdot Float_r + One_r v \cdot BitMap_r$$

then the input description for service Z would constitute a match and provided that the other sub-features matched sufficiently, including its vector encoded QoS_v, then the service could become activated. Note that a different service having exactly one float and bitmap input, would better match the input specification.

2) Building the Quality of Service Vector: For QoS, we employ a slightly different encoding scheme. For example, a QoS metric often has a requirement to meet a certain a minimum or maximum value for the metric in question. An example of a static QoS metric might be that the service must possess a minimum of four CPU cores. A simple way to encode minimum or maximums, such that they are semantically comparable is in the form of a bag of acceptable values. In the number of cores example, to specify four or more cores we can encode:

$$CpuCores_r \cdot (Four_r + Five_r + Six_r + \ldots + MaxCores_r)$$

Therefore, an individual service that encodes its *CpuCores* QoS as $CpuCores_r \cdot Four_r$ or $CpuCores_r \cdot Eight_r$ would be a match.

[1]Currently, we can build vector representations using JSON or XML description of services.

For time-critical distributed applications, *available compute power* might be a better QoS and could be a normalised value combining number of CPUs and GPUs, memory, clock-speed along with the current load. In order to facilitate semantic comparisons of such a metric, a service calculating its value would then quantise it to the nearest higher or lower value depending on if the expected comparison is a max or min requirement. Dynamic QoS metrics such as *battery life percentage* or *available runtime* can also be encoded in this way. For example, aggregate bandwidth, obtained by each service actively monitoring its local bandwidth with pings, might be quantised to $(1Kb, 10Kb, 100Kb, 1Mb, 10Mb, 100Mb, 1Gb)/Sec$. Such values are then converted to an enumeration thereby allowing us to encode ranges that represent different underlying values with the same role vectors. Thus, a minimum bandwidth QoS requirement of, say, 100Mb/sec (100Mb is in 6th position in the above list) would be encoded as follows:

$$Bandwidth_r \cdot (Six_r + Seven_r)$$

The above describes our current method for encoding service descriptions and QoS as BSC vectors. VSA superposition allows us to combine any set of individual functional and QoS parameters into a bag of features for simultaneous comparison and matching enabling a far more flexible approach than the ontology-style approach. Both [19] and [20] describe methods that can be used to combine multiple QoS metrics into a single normalised value that reflects the weighting given to each individual metric. We are currently investigating how best to encode this type of metric using our VSA representation.

IV. Describing Workflows using Vector Symbolic Architecture

As discussed in Section II, VSAs employ two operations—i.e., *binding* and *superposition*. *Binding* is used to build *role-filler* pairs which allow sub-feature vectors to remain separate and identifiable (although hidden) when bundled into a compound vector via *superposition*. For BSCs, *binding* is a lossless operation, while *superposition* is lossy. Kleyko [10, Paper B, page 80] supplies a mathematical analysis of the capacity of a single compound vector such that it can be reliably unbound, i.e., its sub-feature vectors can be reliably detected within the compound vector. This analysis shows that for 10kbit binary vectors the upper limit of superposition is 89 sub-vectors. To encode large workflows with more complex service descriptions we require a method for combining more vectors into a single vector whilst maintaining the semantic matching properties.

Chunking is a recursive binding method that combines groups of vectors into a single compound vector. The resultant vectors are then used as the basis for further chunking operations, thus, recursively producing a hierarchical tree structure as shown in Figure 1. Chunking proceeds from the bottom up so that each node in the tree is a compound vector encapsulating the child nodes from the level below. Various methods of recursive *chunking* have been described [7, 9, 10, 15]. However, such methods suffer from limitations when employed for multilevel recursion: some lose their semantic matching ability even if only a single term differs, others cannot maintain separation of sub-features for higher level compound vectors when lower level chunks contain the same vectors [9, page 148] [7, pages 61, 72, 74–para2] [10, Encoding Sequences, page 14]. We addressed these issues and describe a novel recursive encoding scheme that provides semantic matching at each level by combining two different methods of permuting vectors.

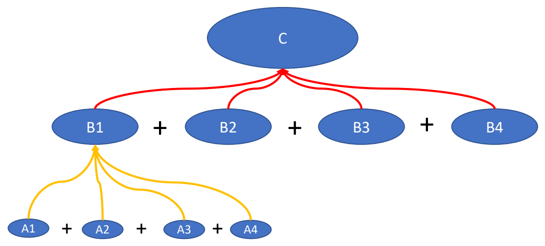

Fig. 1: Vector Chunk Tree

In our scheme, the terminal nodes are worker services, the higher level nodes are concepts used to apply grouping to parts of the workflow. The chunking process occurs from the bottom up so that the bottom level nodes, $\{A1, A2, A3, \ldots\}$ are combined via a functional partitioning scheme into a concept node, e.g., $B1$. These higher level nodes—referred to as *clean-up memory* [7, 9, 10]—are still services but they simply provide a proxy to the worker services to be unbounded and executed; thus, they are typically co-located with the first service of the sub-sequence they represent, e.g., $B1$ can reside on the same compute node as the $A1$ service and hence, when $B1$ becomes activated, there is no need for a network transmission in order for $B1$ to activate its first worker service, $A1$. In a centralized system, *Clean-up memory* is typically implemented as an auto-associative memory. For our distributed workflow system, clean-up memory is implemented by the services themselves, which are distributed throughout the network, matching and resolving to their own vector representations.

Recchia and Kanerva point out that for large random vectors, any mapping that permutes the elements can be used as a binding operator, including cyclic-shift [14]. The encoding scheme shown in 8 employs both XOR and cyclic-shift binding to enable recursive bindings capable of encoding many thousands of sub-feature vectors even when there are repetitions and similarities between sub-features:

$$Z_x = \sum_{i=1}^{cx} Z_i^i \cdot \prod_{j=0}^{i-1} p_j^0 + StopVec \cdot \prod_{j=0}^{i} p_j^0 \tag{8}$$

Omitting $StopVec$ for readability, this expands to,

$$Z_x = p_0^0 \cdot Z_1^1 + p_0^0 \cdot p_1^0 \cdot Z_2^2 + p_0^0 \cdot p_1^0 \cdot p_2^0 \cdot Z_3^3 + \ldots \tag{9}$$

Where

- $\cdot$ is defined as the XOR operator;
- $+$ is defined as the Bitwise_Majority_Vote/Add operator;
- The exponentiation operator is redefined to mean cyclic-shift—i.e., positive exponents mean C_{shift_right}, negative exponents mean C_{shift_left}. Note that cyclic shift is key to the recursive binding scheme since it distributes over $+$

(i.e., bitwise majority addition) and $\cdot$ (i.e., XOR) hence it automatically promotes its contents into a new part of the hyper-dimensional space; thus, keeping levels in the chunk hierarchy separate;

- Z_x is the next highest semantic *chunk* item containing a *superposition* of x sub-feature vectors. Z_x chunks can be combined using 8 into higher level chunks. For example, Z_x might be the superposition of $B1 = \{A1, A2, A3, \ldots\}$ or $C = \{B1, B2, B3, \ldots\}$;
- $\{Z_1, Z_2, Z_3, \ldots Z_n\}$ are the sub-feature vectors being combined for the individual nodes of Figure 1. Each Z_n itself can be a compound vector representing a sub-workflow or a complex vector description for an individual service step, built using the methods described in Section III;
- $p_0, p_1, p_2, \ldots$ are a set of known *atomic* role vectors used to define the current position or step in the workflow.
- cx is the chunk size of vector Z_x, i.e., the number of sub-feature vectors being combined; and
- $StopVec$ is a role vector owned by each Z_x that enables it to detected when all of the steps in its (sub)workflow have been executed.

A. Ordered Unbinding of High-level Concept Vectors

Eq.(8) is used recursively to build a workflow request, conceptually creating a hierarchical chunk tree as shown in Figure 1. The resulting output is a set of VSA vectors representing the non-terminal nodes, $\{C, B1, B2, B3, \ldots\}$, each of which is a single VSA vector, Z_x that is itself a compound vector representing the ordered sequence of its own children.

The generalized version of the concept vectors, $\{C, B1, B2, \ldots\}$, is shown in (9). Note that, in this form every sub-step Z_n is permuted by at least one p vector which effectively hides each Z_n (the p vector permutation ensures that each combined roll-filler pair is orthogonal to the 'self-description' vectors built by each VSA service listening for work on the network). The workflow is discovered and orchestrated on the distributed services by, essentially, repeatedly *unbinding* the workflow vector, using 10 or 12, before re transmitting it to the network.

Referring to Figure 1, control first passes down the chunk tree, i.e., from $C \rightarrow B1 \rightarrow A1$ using 10, before traversing horizontally, $A1 \rightarrow A2 \rightarrow A3 \rightarrow A4 \rightarrow B1\_StopVec$) via 12. At this point $B1$ *sees* its $StopVec$ and employs 12 to activate $B2$ which then activates its sub-workflow via 10 and so forth.

Starting a (sub)workflow:

$$Z_1' = (p_0^0 . (T + Z_x))^{-1} \tag{10}$$

$$Z_1' = p_0^{-1} . T^{-1} + \boxed{Z_1^0} + p_1^{-1} . Z_2^1 + p_1^{-1} . p_2^{-1} . Z_3^2 + \ldots \tag{11}$$

Traversing horizontally:

$$Z_{n+1}' = (\mathbf{p_n^{-n}} \ . \ \mathbf{Z_n'})^{-1} \tag{12}$$

$$Z_2' = (p_1^{-1} . Z_1')^{-1} = p_1^{-1} . p_0^{-2} . T^{-2} + p_1^{-1} . Z_1^{-1} + \boxed{Z_2^0} + p_2^{-2} . Z_3^1 + \tag{13}$$

When starting a (sub)workflow using (10) notice that Z_1 has been exposed, as shown in (11). That is, Z_1' is effectively a noisy copy of the currently required workflow step, Z_1, while at the same time it is also a 'masked' description of the full (sub)workflow request. Thus, listening services can only match to Z_1' if they are semantically similar to Z_1. Note that, the act of matching gives a service no other information; for example, it cannot deduce by matching alone whether the match occurred at step 1 or step 30 of the workflow. Hence, the introduction of the T vector in 10 which is used to enable calculation of a node's position within the workflow.

The T vector is a known *atomic* role vector. It is added to a high level node's, clean, (sub)workflow vector, Z_x, before the node uses (12) to expose its first workflow step for transmission to the network. We note that, (10) is just a special version of (12). Notice in (11) and (13), how the $\boldsymbol{T}$ vector becomes permuted in a predictable way. Once the currently active service has completed its own workflow step it uses the current permutation of the $\boldsymbol{T}$ vector to calculate its position $\boldsymbol{n}$ within the received request vector. It can then activate the next workflow step in the request by repeating the *unbind* operation on the request vector, generalized in 12. Thus, the workflow proceeds in a completely decentralized manner whereby each node is activated when its preceding node, or parent, *unbinds* the currently active chunk vector, creating the next request vector, which it then multicasts to the network for matching and processing.

B. Pre-provisoning and Learning to Get Ready

From 12 we see that each workflow step is exposed by iterative application of p vector permutations. Non-matching services can use this method to *peek* a vector enabling anticipatory behavior such as the pre-provisioning of a large data-set or changing a device's physical position (e.g., drones). Obviously, services can *peek* multiple steps into the future and could *learn* how early to start pre-provisioning. This ability to anticipate could be used to perform more complex, on-line, utility optimization learning. For example, a drone monitoring multiple workflows may be able to understand that it will be needed in 10 minutes to perform a low priority task and in 15 minutes for a high priority task. Under these circumstances it may choose not to accept the low priority task.

C. Alternate to QoE for Distributed Transient Environments

As can be seen in 11 and 13, when a particular workflow step is exposed for discovery and execution by unbinding, it is 'surrounded by' (i.e. it is in superposition with) the rest of the workflow steps which, as can be seen, are permuted in a specific way depending on the position of currently exposed/active service in the workflow. We can think of this as the current permutation of the workflow vector and it constitutes a context for the workflow step currently in focus. We are investigating the use of these contexts as an analog of QoE. The idea is that when a services successfully participates in a workflow it will remember the permutation state of the workflow vector via which it was activated. If a particular service successfully participates in the same workflow repeatedly; these workflow context memories can be used to increase the particular service's utility with respect to the specific workflow. We suggest that this might be an interesting analog of traditional QoE measures which we believe these will be almost impossible to measure and collect in distributed transient environments. The idea is that a service that often helps complete a particular workflow should be seen as a more valuable partner by the other service steps, hence an analog of QoE.

V. Test Cases

In order to demonstrate the applicability and scalability of our encoding scheme, we provide two use cases where we have applied the VSA system to both linear and complex workflows. We also provide an experimental evaluation for the correctness and scalability of the proposed approach.

A. Dynamic Mode Linear Workflow

To compare with alternative approaches such as those described in [21], we have semantically encoded the entire text of Shakespeare's play Hamlet into this type of hierarchical semantic vector representation. In this example the component services at the lowest level are the 4620 unique words of the play; the semantic level above are the individual stanzas spoken by each character (not shown in the diagram); the level above this are individual scenes of the play(e.g., A1S1, A1S2); next are the five acts, A1-A5 and then finally a single 10kbit vector semantically represents the whole play (Hamlet). A

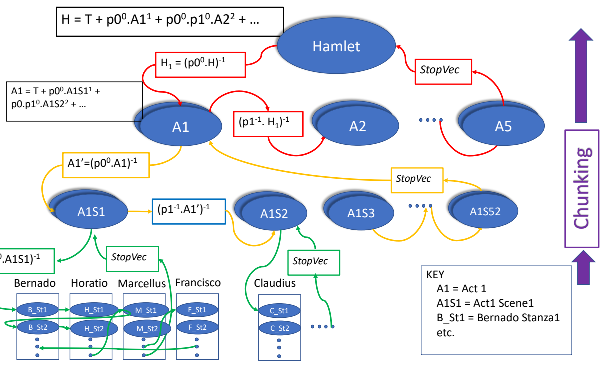

Fig. 2: Hamlet as a serial workflow

vector alphabet, a unique vector per alphabet character, was used to build compound vectors for each word-service in the play. The idea is that each letter making up a word represents some feature of a service description, i.e., analogous to the different input/output/name/descriptions parts of a real world service. Thus, variable lengths of words and similarity of spellings represent a mix of different services of different complexity and functional compatibility. At the next higher level sentences represent a more complex sub-workflow, and so on. It is important to recognise that the higher level vectors do semantically represent the recursive binding from all the levels below. We use this fact to allow alternative, semantically similar, service compositions to be invoked if the best matching composition is not available. Figure 3 shows the word service *where* being invoked as an alternate to *there* which was unavailable. Note also, that when *where* completes it automatically re-synchronises to the original worflow because it simply *unbinds* the next step from the original workflow vector it received.

To simulate QoS matching we employed two random variables representing *current load* and *battery life*. From a requesters point the idea was that *current load* should be minimized and *battery life* should be maximized so that we could try out our min/max idea for encoding QoS as described in Section III-2. Acceptable ranges of values were randomly chosen when encoding service request vectors and each services simulated its own QoS in the same manner. Thus matching on functional as well as QoS criterion was tested. Multiple copies of the individual component vectors at each level are distributed in a communications network as services and by multicasting the top level vector the whole play is performed in a distributed manner with 29,770 component word services being invoked in the correct order.

B. Static Mode Complex Workflow

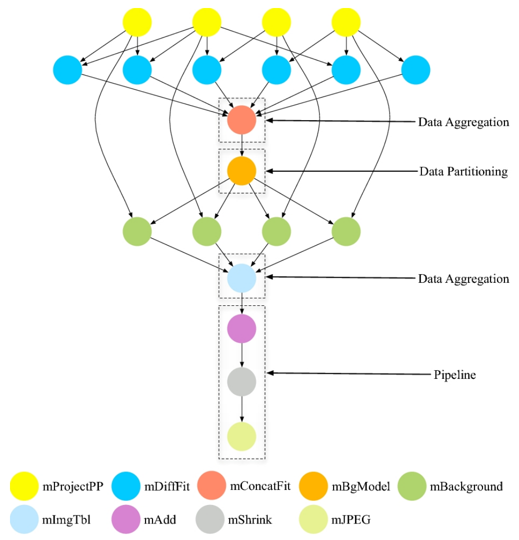

Fig. 3: Montage Workflow

We note that the Hamlet workflow is a linear sequence of services, next we show how such an approach can be extended to DAG workflows by employing three phases:

1) A recruitment phase where services are discovered, selected and uniquely named.
2) A connection phase where the selected services connect themselves together using the newly generated names.
3) An atomic *start* command indicates to the connected services that the workflow is fully composed and can be started.

Thus, in mathematical terms, using (9):

$$\begin{aligned} WP\ &= p0^0.(Recruit_{Nodes})^1 + \\ &\quad p0^0.p1^0.(Connect_{Nodes})^2 \quad + \quad p0^0.p1^0.p2^0.Start^3 \\ Recruit_{Nodes} &= p0^0.Z_1^1 + p0^0.p1^0.Z_1^2 + ...p0^0.p1^0.p2^0.p3^0.Z_1^4 \\ &\quad + p0^0.p1^0.p2^0..\ .p4^0.Z_2^5\ ... +\ p0^0.p1^0.p2^0..\ .p9^0.Z_2^{10} \\ &\quad + p0^0.p1^0.p2^0..\ .p10^0.Z_3^{11}\ +\ p0^0.p1^0.p2^0..\ .p11^0.Z_4^{12} \end{aligned}$$

$$
\begin{aligned}
&+p0^0.p1^0.p2^0..\ .p12^0.Z_5^{13} \ + \ p0^0.p1^0.p2^0..\ .p15^0.Z_5^{16} \\
&+p0^0.p1^0.p2^0..\ .p16^0.Z_6^{17} \ + \ p0^0.p1^0.p2^0..\ .p17^0.Z_7^{18} \\
&+p0^0.p1^0.p2^0..\ .p18^0.Z_8^{19} \ + \ p0^0.p1^0.p2^0..\ .p19^0.Z_9^{20}
\end{aligned}
$$

$$
\begin{aligned}
Connect_{Nodes} = &\left(p0^0.\mathbb{P}_1^1 + \ p0^0.p_1^0.\mathbb{C}_1^2\right) + \\
&\left(p0^0.p1^0.p2^0.\mathbb{P}_2^3 + p0^0.p1^0.p2^0.p3^0.\mathbb{C}_2^4\right) + ...
\end{aligned}
$$

where each Z_n in $Recruit_{Nodes}$ is the semantic/compound vector representation of each service, built from the <job> entries found in the DAX. A generic description of each service was built from the service name and its description and used to build the workflow request vector. For individual instances of a service, e.g., *mDiffFit*, we additionally encode the instance's parameters and resource names to create similar but distinct service instances to, again, show that service discovery can be achieved when descriptions are not identical.

The resulting workflow, WP, is a superposition representing the linear sequence of steps needed to discover, connect and initiate the workflow. Hence, execution of the workflow proceeds in a similar manner to that described in Section IV-A, but with some additional workflow specific processing carried out by each selected node. The top-level vector, *WP* is prepared as per (10)

$$
WP_1 = (T + p_0^0.WP)^{-1} = Recruit_{nodes} + noise
$$

When multicast, this exposes and activates the $Recruit_{nodes}$ service which, operating as a cleanup service, carries out the the same operation to initiate the recruitment phase.

$$
\begin{aligned}
Recruit'_{nodes} &= (T + p_0^0.Recruit_{nodes})^{-1} \\
R'_1 &= p_0^{-1}.T^{-1} + Z_1^0 + p_1^{-1}.Z_1^1 + p_1^{-1}.p_2^{-1}.Z_1^2 + ...
\end{aligned}
$$

where Z_1 is a request for the first node in the DAG, an mProjectPP in the Montage DAG. This will be matched by all listening mProjectPPs. Acting as the local arbitrator, see Section **??**, the $Recruit_{nodes}$ service multicasts its preferred match from the replies received. The newly discovered and activated service uses the current permutation of the T vector to calculate its position ($NODE_{id}^n$) in the $Recruit_{nodes}$ phase from which it can calculate its unique parent and child vector names to be used during the $Connect_{Nodes}$ phase. Thus, the first mProjectPP, having position *p0* and being a Z_1, calculates its parent and child names as,

$$
\mathscr{P}_0 = \ Z_1^0 \ . \ \left(NODE_{id}^0 \ . \ ROLE\_parent\right)
$$
$$
\mathscr{C}_0 = \ Z_1^0 \ . \ \left(NODE_{id}^0 \ . \ ROLE\_child\right)
$$

It then enters *Listening for Connections Mode* while, as the new *local arbitrator*, it also multicasts the next recruitment request by performing an unbind on its received vector R'_1, thus $R'_2 = (p_1^{-1}.Z'_1)^{-1} = p_1^{-1}.p_0^{-2}.T^{-2} + p_1^{-1}.Z_1^{-1} + Z_1^0 + p_2^{-2}.Z_1^1 + ...$ which will cause another mProjectPP to be selected and this decentralized process repeats until the last service to be recruited, the Z_9, mjPeg, service unbinds and transmits the next vector, the $Recruit_{nodes}$ *StopVec*. The $Recruit_{nodes}$ cleanup service detects its stop vector, causing it to perform an unbind and multicast of WP' therby activating the $Connect_{nodes}$ phase:

$$
WP_2 = (T + p_1^{-1}.WP_1)^{-1} = Connect_{nodes} + noise
$$

At this point, all recruited services are listening for connection request on their unique parent and child vectors. The activated $Connect_{nodes}$ service, acting as a cleanup service, uses (10) to initiate and activate the first *parent* node of the $Connect_{nodes}$ phase.

$$
\begin{aligned}
Connect'_{nodes} &= (T + p_0^0.Connect_{nodes})^{-1} \\
\mathbb{P}'_1 &= p_0^{-1}.T^{-1} + \mathbb{P}_1^0 + p_1^{-1}.\mathbb{C}_1^1 + p_1^{-1}.p_2^{-1}.\mathbb{P}_2^2 + ...
\end{aligned}
$$

When a service matches to its *parent* vector it simply performs the next unbind/multicast since in doing so it will activate its associated *child* service, automatically informing the child service of the location of its resources/output/ip-address.

When a service receives a multicast that matches to its *child* vector it can lookup the sender/parent's ip-address and send a unicast '*hello*' message to the parent, thus establishing the required connection before activating the next *parent* by performing a further unbind/multicast of the $Connect_{nodes}$ vector. This process repeats until the final child request is processed causing the $Connect_{nodes}$ service to detect its *StopVec* which, in turn, causes it to unbind and multicast the *StartVec* indicating to all nodes that the workflow has been fully constructed and processing can be started.

VI. Evaluation

Our evaluation of the scalability of the VSA approach for linear workflows has already been presented using the Hamlet example in Section V-A. The evaluation was performed using the CORE/EMANE network emulator to simulate a MANET network and used a MANET multicast routing protocol to communicate vectors between the nodes containing the services. Multicasting the top level Hamlet vector results in the whole play being enacted by worker services that generate each word in the play. The VSA workflow implementation of Hamlet has a number of advantages over the Newt[21] implementation. Specifically the Newt implementation requires that the IP address of participating services be known and encoded into the workflow, whereas, our VSA approach can discover the service(word/sentence) needed on the fly using semantic matching. In Newt, if the service specified by IP address becomes unavailable; i.e., we intentionally move it out of wireless range in CORE, then the workflow halts and is broken. In VSA Hamlet, the same action results in the automatic discovery of multiple exact, and near-match candidate word/sentence/services and the best match is then chosen. When multiple functionally equal matches were discovered the *Local Arbitration* function ensured that the service having best simulated utility was chosen and logged as such. The best *'near'* match was chosen when we contrived to make exact matches unavailable in CORE. Additionally, the advantage of passing around the workflow as a vector superposition was highlighted because the stand-in service automatically resynchronized the workflow after *'speaking'* its substitute word by simply performing an unbind and transmit of the workflow vector it received. Newt has none of these capabilities.

Our evaluation of the more complex workflows was aimed at the following: to demonstrate that complex workflows could be automatically encoded into a symbolic vector representation and then recursively decoded to assemble the required work in a decentralized setting; to show that the workflow constructed was also resilient to changes in the communications network; and to demonstrate that services with the highest utility could be identified and selected using the semantic matching mechanism.

We again ran a series of experiments using the CORE/EMANE network emulator to simulate a MANET network. Pegasus DAX workflows were processed using the VSA creator to build the semantic vector workflow encodings and also to generate the service description vectors that semantically describe each of the component services. Multiple copies of the component services were randomly distributed on the network nodes. Each service was enabled with our VSA cognitive layer containing the appropriate semantic vector for that service. The workflow request vector was launched from some node in the network and the workflow was constructed in a decentralized manner, with control being passed between services as the workflow vector was recursively unbound. During the execution of the process we extracted a range of metrics that provided a detailed log of the run and the order of execution. Using this log we created a graph of the set of nodes and edges that were selected and we used Graphviz to show the result. Figure **??** shows the results for the five different Pegasus workflows we evaluated. The coloured images represent the Pegasus generated workflows and blue workflows show the VSA generated reconstruction of the workflows. Aside from the cosmetic difference, this demonstrates that all workflows were composed and correctly connected accurately in all cases.

To demonstrate the resilience of the approach, we modified the network connectivity to demonstrate that different instances of the correct services were selected and that this still produced the same required workflow. We also demonstrated that if the same services had different QoS utility measures that the services with the higher utility were selected in preference to those of lower utility.

VII. Conclusions And Future Work

In this paper, we addressed the complex problem of how to represent and enact decentralized time-critical applications. Specifically we investigated how data analytics tasks, formulated as complex workflows, could operate in dynamic wireless networks, without any central point of control. To this end, we described an architecture that exposes a cognitive layer by using a Vector Symbolic Architecture (VSA) to extend services with semantic service descriptions and time-critical constraints required to specify the QoS/QoE. We demonstrated the viability of this approach by showing an empirical evaluation that such VSA encoding methods work and are scalable. We then described the architecture of our approach and the components it provides to enable decentralized fitness functions for on demand resource discovery and allocation.

We demonstrated that our approach can encode workflows containing multiple coordinated sub-workflows in a way that allows the workflow logic to be unbound on-the-fly and executed in a completely decentralized way. We showed that time-critical QoS and QoE metrics for each workflow, sub-workflow or even service can be encoded into a single vector that provides an extremely compact (10kbits) common workflow format exchange for a MANET, which can be passed around using standard group transport protocols (e.g., multicast). We also showed that semantic comparisons can be made at each level of the architecture to support scoped searching and that the scheme is extensible—i.e., new parameter or constraint can be plugged in and encoded to address practically any real-world scenario.

In the future, we will investigate different schemes for discovery and matchmaking, which are capable of supporting different modes of use. For example, we are currently looking at using the look ahead *peeking* capability of VSAs in combination with proactive announcements that will be capable of pushing utility metrics calculations to the client that need to consume them. A key element of future work is to investigate alternative ways to encode semantics and to measure the semantic similarity of services and their QoS and QoE. We are investigating methods that capture the previous contexts, including QoS metrics, in which a particular workflow has operated in, as well as other methods that avoid rigid ontology style approaches. For time-critical applications in MANET environments we are investigating alternatives to local arbitration that will allow the fittest service to rapidly emerge from a group of compatible competing individual services. Using symbolic vectors to semantically represent services and workflows enables suggested alternative service compositions to be automatically generated when component services of an existing workflow are missing or cannot be accessed. We are investigating if viable alternative compositions can be generated and automatically validated using this approach.

VIII. Acknowledgements

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, “Web services composition: A decades overview,” *Information Sciences*, vol. 280, pp. 218–238, 2014.

[2] S. Corson and J. Macker, “Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and

Evaluation Considerations," RFC 2501 (Informational), Internet Engineering Task Force, Jan. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2501.txt

[3] S. Basagni, M. Conti, S. Giordano, and I. Stojmenović, *Mobile Ad Hoc Networking: Edited by Stefano Basagni...[et Al.]*. IEEE, 2004.

[4] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*. New York, NY, USA: ACM, 1998, pp. 85–97.

[5] T. Pham, G. Cirincione, A. Swami, G. Pearson, and C. Williams, "Distributed analytics and information science," in *Information Fusion (Fusion), 2015 18th International Conference on*. IEEE, 2015, pp. 245–252.

[6] D. Verma, G. Bent, and I. Taylor, "Towards a distributed federated brain architecture using cognitive iot devices," in *9th International Conference on Advanced Cognitive Technologies and Applications (COGNITIVE 17)*, 2017.

[7] T. A. Plate, *Distributed representations and nested compositional structure*. University of Toronto, Department of Computer Science, 1994.

[8] R. W. Gayler, "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," *arXiv preprint cs/0412059*, 2004.

[9] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors." *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/cogcom/cogcom1.html#Kanerva09

[10] D. Kleyko, "Pattern recognition with vector symbolic architectures," Ph.D. dissertation, Luleå tekniska universitet, 2016.

[11] G. E. Hinton, "Mapping part-whole hierarchies into connectionist networks," *Artificial Intelligence*, vol. 46, no. 1-2, pp. 47–75, 1990.

[12] M. N. Jones and D. J. K. Mewhort, "Representing word meaning and order information in a composite holographic lexicon," *psychological Review*, vol. 114, no. 1, pp. 1–37, 2007.

[13] G. E. Cox, G. Kachergis, G. Recchia, and M. N. Jones, "Toward a scalable holographic word-form representation," *Behavior research methods*, vol. 43, no. 3, pp. 602–615, 2011.

[14] G. Recchia, M. Sahlgren, P. Kanerva, and M. N. Jones, "Encoding sequential information in semantic space models: comparing holographic reduced representation and random permutation," *Computational intelligence and neuroscience*, vol. 2015, p. 58, 2015.

[15] T. A. Plate, *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*. Stanford, CA, USA: CSLI Publications, 2003.

[16] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *Science*, vol. 338, no. 6111, pp. 1202–1205, Nov. 2012. [Online]. Available: http://www.sciencemag.org/content/338/6111/1202

[17] G. B. G. d. M. R. G. Christopher Simpkin, et Ian Taylor, "Constructing distributed time-critical applications using cognitive enabled services," *Future Generation Computer Systems*, vol. 100, pp. 70–85, 2019.

[18] C. Simpkin, I. Taylor, G. A. Bent, G. de Mel, and R. K. Ganti, "A scalable vector symbolic architecture approach for decentralized workflows," in *COLLA 2018, The Eighth International Conference on Advanced Collaborative Networks, Systems and Applications(COLLA),*, 2018.

[19] Y. Liu, A. H. Ngu, and L. Z. Zeng, "Qos computation and policing in dynamic web service selection," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM, 2004, pp. 66–73.

[20] E. Vinek, P. P. Beran, and E. Schikuta, "A dynamic multi-objective optimization framework for selecting distributed deployments in a heterogeneous environment," *Procedia Computer Science*, vol. 4, pp. 166–175, 2011.

[21] J. P. Macker and I. Taylor, "Orchestration and analysis of decentralized workflows within heterogeneous networking infrastructures," *Future Generation Computer Systems*, 2017.