An Algorithm for Load Balancing in Network Management Applications Managing Virtualised Network Functions

Sajeevan Achuthan
Network Management Lab,
Ericsson,
Athlone, Co. Westmeath,
Ireland
sajeevan.achuthan@ericsson.com

John Keeney
Network Management Lab,
Ericsson,
Athlone, Co. Westmeath,
Ireland
john.keeney@ericsson.com

Liam Fallon
Network Management Lab,
Ericsson,
Athlone, Co. Westmeath,
Ireland
liam.fallon@ericsson.com

***Abstract*—In network management applications, load balancing has typically been achieved by manually configuring the application in-situ or using configuration information generated using offline tools. As networks increase in scale and heterogeneity, management applications are being designed to take advantage of scalable computer hardware and often have multiple instances, each of which bears a portion of the management load of the network elements or functions being managed. The communication between management applications and the network elements they are managing is often stateful, posing a challenge for load balancing because connections must be managed if responsibility for managing some network elements change from one instance to another during load balancing. In this paper, we describe an algorithm that enables load balancing of telecommunication management applications for an arbitrary number of VNFs (Virtual Network Functions). The algorithm, triggered by NFV (Network Function Virtualisaton) orchestration, is horizontally scalable and there is no single point of failure once two or more management application instances are running.**

I. Introduction

The time honoured approach for load balancing in network management applications is to configure the parameters exposed by the application offline using guidelines provided by the application provider. This was an appropriate approach in GSM systems where the number of network elements was of the order of hundreds of network elements. In recent years, there has been an order of magnitude increase in the number of network elements being managed by mobile network management systems; in LTE, networks with hundreds of thousands of network elements are not uncommon. Networks are increasingly heterogeneous and may be running four or more radio technologies (such as 2G, WCDMA, LTE, Wireless LAN), and may have very complex radio architectures with macro, micro, pico, and even femto cells [1], and may be deployed as bespoke nodes or using Network Function Virtualization [2].

In parallel, the computers on which management systems run have evolved from systems with small numbers of monolithic machines to systems where many computing entities run in a highly distributed manner, often deployed across multiple machines on blade systems, or increasingly as virtual machines operating in virtualised environments. Such distributed infrastructures often have support for seamlessly adding and removing computing resources, whether by changing the physical or virtual machines, or by adding/removing machines on the fly[3]. The advent of this heterogeneity, the increase in scale in managed networks, and the distribution of management applications makes offline tuning of application load balancing impractical.

Consider a management application that is receiving and processing streamed events from managed network elements running in a network deployed in NFV clusters. The application runs a number of instances in order to handle the volume of events being produced by the network, each of which is deployed on the distributed computing platform. Each instance receives events from a configured subset of the network elements in the managed network. When a managed network element is added to or removed from the network by NFV orchestration, the configuration of a management instance is amended and that instance may need to be re-initialized. Likewise, when the cloud infrastructure on which the management system is running adjusts the number of management application instances or changes the resources available to the management application, the configuration of a number of instances in the distributed application is amended and each of those instances must be re-initialized.

In this paper, we describe an approach to enable load balancing of VNFs of a telecommunication network across an arbitrary number of distributed processing instances that make up a management application. The approach is horizontally scalable and there is no single point of failure once two or more processing instances are running.

We have applied this algorithm in the collection and processing of events from network elements. In such an application, network elements (which may be running as VNFs) may be added to or removed from the network at any time and many instances of a distributed management system are used to collect and process events. The number of management application instances running at any given time varies based

on the current network size, the current event load, or due to particular application instances failing. The load balancing mechanism is used to balance the management load fairly across all running application instances. New management instances are added or removed to handle periods of heavy or light load.

Our algorithm can be applied to any management application where state must be managed across distributed instances.

II. Background and Related Work

Management systems have evolved to use distributed architectures that allow the processing power of a distributed system of cooperating computing entities to be harnessed to manage large modern heterogeneous mobile networks. Such architectures are designed to allow management applications to scale horizontally; many instances of the application are run in parallel in separate processes and threads with each instance carrying the load of a small portion of the managed network. In this way, increasing or decreasing the number of instances of an application that is running at any given moment can adjust the capacity of an application. The network load is distributed in such a management application using methods such as dividing the bespoke and virtualised network elements evenly across all instances or by measuring the load being produced by each network element and planning distribution based on those measurements.

Current approaches for load balancing in distributed computing (cloud) platforms focus on providing support in the distributed computing infrastructure that allows applications to elastically increase and decrease the number of instances they are running [4][5], or to add or remove resources available to already running instances. Such approaches do not consider handling of shared state that applications may have, that applications may have interrelationships that must be managed when instance counts change, or that load may need to be transferred between instances.

In contrast to most stateless distributed/cloud architectures such as RESTful web services [6], management applications generally use stateful session semantics to interact with network elements. A management application establishes a management session with a network element, carries out some operations, and then closes the session. Such sessions may be used to collect statistical information [7], carry out configuration operations [8] or receive event streams [9]. Such management sessions, particularly those used for alarm or event collection, are often very long lived; often lasting weeks or even months if no change is needed. A particular instance of a management application using such a stateful long-lived session towards a particular network element must manage the session state for the duration of that session. It is, therefore, inherently difficult to exploit flexible distributed architectures such as cloud platforms for load balancing in network management applications because network element session state must be managed when the number of management instances change. In order to change the allocation of a network element from one management application instance to another, the management session between the network element and the *old* management instance must be torn down and a management session must be established between the managed element and the *new* management instance.

TABLE I
Comparison of Approaches to Load Balancing

	Manual	**Centralised**	**Automatic**
Planned	Yes	Yes	No
Central Control	Yes	Yes	No
Co-Ordination	Human	Central Application	Autonomic
Adaptive	No	No	Yes
Reset in Deployment	Yes	Yes	No

In existing solutions, network elements are either manually allocated to management application instances, or are allocated using centralised algorithms. To ensure that management sessions with network elements are properly handled, each amended management application instance must be shut down, its new configuration must be set, and the management application instance must be restarted. It is, therefore, difficult to automate redistribution of load because shutdown of management instances must be coordinated across all amended network elements. There are a number of issues with this approach: configuration is centrally planned and pushed to network elements; redistribution must be coordinated and executed as a single operation across all concerned network elements; redistribution of load is static; complex load balancing algorithms, such as those based on the current load on management application instances are difficult to apply (round-robin allocation are more typical); and, if an instance of a management application fails, the load being carried by that instance is not automatically reallocated to other instances.

Table I presents an assessment of Manual, Centralised, and Automatic approaches to load balancing. While manual and centralised approaches allow planning and central control, we chose an automatic approach because it provides autonomic co-ordination, adaptiveness to changes in the NFV infrastructure, and eliminates of the need to carry out resets on existing NFV instances during the load balancing procedure.

ETSI has introduced the NFV (Network Function Virtualisation) architectural framework [2] shown in Fig.1. This framework allows virtual network functions to run on commodity IT hardware and networks. VNFs can be spun up and down by the orchestrator on demand as network conditions vary. The introduction of NFV facilitates load balancing in network management applications because the Orchestrator is a single point of contact for determining when nodes deployed as VNFs appear and disappear or when the resources allocated to such nodes change. A load balancing function in a network management application can use the *Os-Ma* interface to determine whether it should load balance its deployment to adjust to the changed network configuration.

III. The Algorithm

This section introduces a general method that allows dynamic load balancing of a distributed management application.

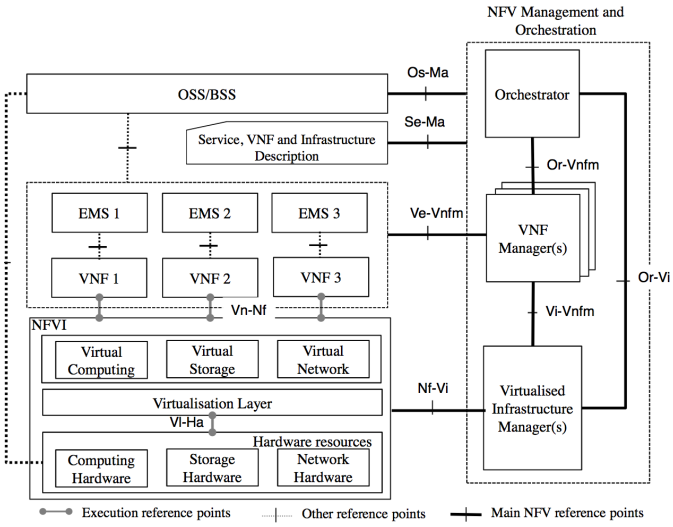

Fig. 1. The ETSI NFV Reference Architectural Framework [2]

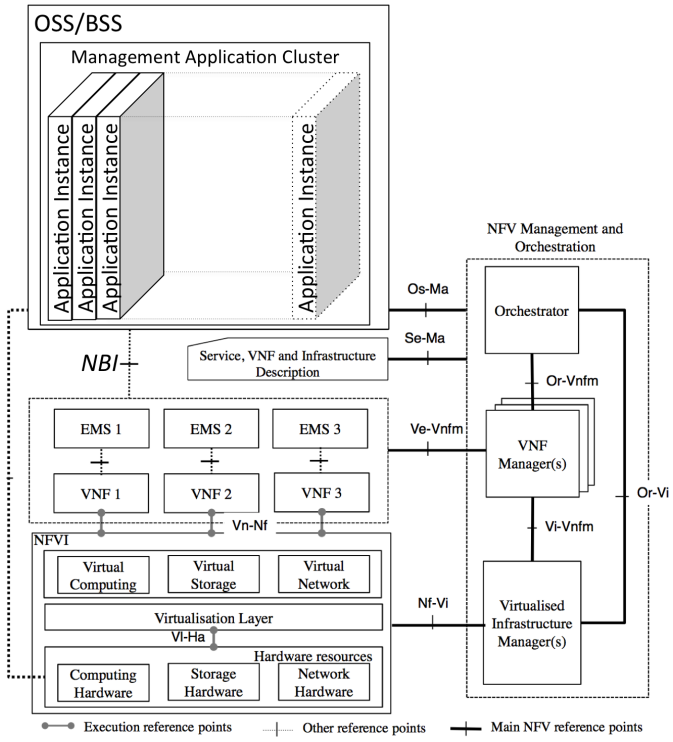

Fig. 2. Distributed Management Application Managing NFVs

The management application runs as a cluster of application instances as shown in Fig.2. The management load is distributed across the application instances. The management application uses the *NBI* (NorthBound Interfaces) of the VNFs for functional communication with virtualised network elements. Operations such as alarm reception or event collection are executed over the NBI, just as they are towards bespoke nodes. The management application uses the *Os-Ma* interface to communicate with the NFV orchestrator. The orchestrator informs the management application when it adds or removes

TABLE II
CLUSTER AND INSTANCE INFORMATION

Cluster Information	
State	Parameters (PC)
running	*5% imbalance deactivate at 40% activate at 80%*

Instance Information			
ID	State	Parameters (PI)	Metrics (MI)
1	*running*	*24GB*	*NE1:100/s,NE2:35/s*
2	*suspended*	*16GB*	*NE3:0/s,NE4:0/s*
...	....	...	...
3	*running*	*32GB*	*NEx:72/s,NEy:68/s*

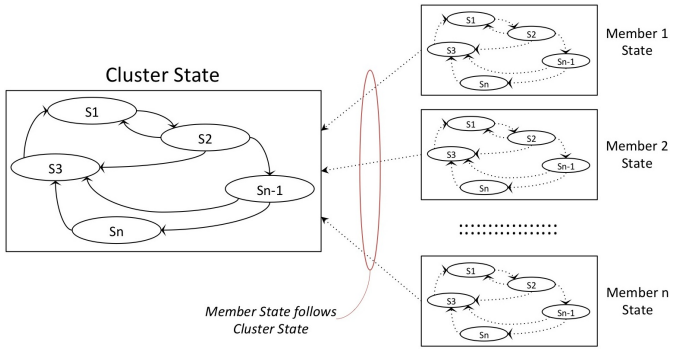

Fig. 3. Cluster State and instance State

virtualised network elements or when it amends the resource allocation to virtualised nodes.

Each running instance of a management application is a member of a distributed application cluster and, in a cluster, all management application instances are peers. The overall state of the application is represented by the overall cluster state, and any common parameters required to trigger load balancing are held as cluster parameters, see Table II. Cluster parameters may include the level of imbalance in the load being borne by instances that may be tolerated before a rebalancing is triggered, the level of load below which instances may be deactivated, or the level of load above which instances may be activated. Each management instance in the cluster also shares its own state, parameters and application metrics with all other instances of the cluster. A record for each management instance is held as shared information, visible to all instances. Examples of instance parameters and metrics are the memory available on an instance and the data rate on the connection between a management instance and its VNFs respectively.

A. State Management

A single cluster instance is nominated *Cluster Master* and assumes the role of monitoring and updating the cluster state. All other cluster instances follow the cluster state as shown in Fig.3. Periodically, the Cluster Master determines whether load balancing should be activated and, if so, it carries out the series of state transitions in cluster state required to execute the load balancing operation using the procedure shown in

Fig.4. Each instance periodically checks if the cluster state has changed and, if so, begins transition to the observed cluster state using the procedure shown in Fig.5. In a new transition, each application instance executes the actions required to change to that new state. In this way, cluster instances *follow* the state changes of the cluster, allowing the state of all cluster instances to be led through the state transitions of the load balancing operation in a decentralised way. Each cluster instance manages its state asynchronously; monitoring the cluster state without any reference to or synchronisation with other instances. As long as the cluster state does not change, cluster instances need not carry out any actions.

The manner in which cluster state influences instance state is shown in Fig.3. The cluster state leads the state of all instances; it is set as the desired next state of all cluster instances.

Consider a set of cluster states S where $\{s_1, s_2, ...s_x, s_y, ...s_n\} \in S$. If the cluster state transitions from $s_2 \rightarrow s_n$, all instances in the cluster will also transition from $s_2 \rightarrow s_n$, executing whatever particular operations are necessary to execute that transition. In normal operation, the cluster state will not transition from $s_x \rightarrow s_y$ until all cluster

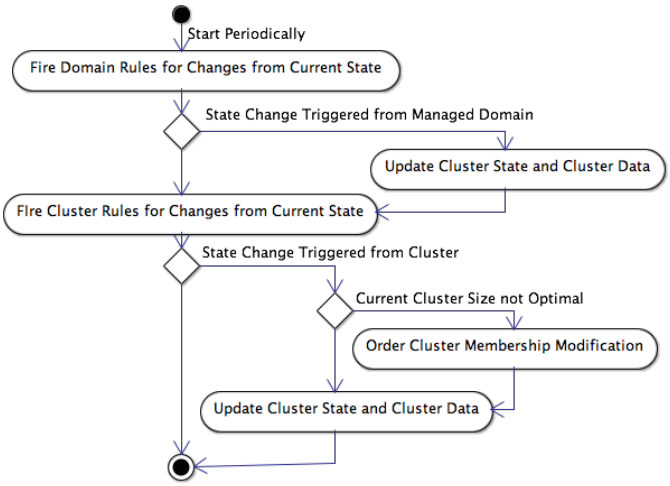

Fig. 4. Cluster State Management

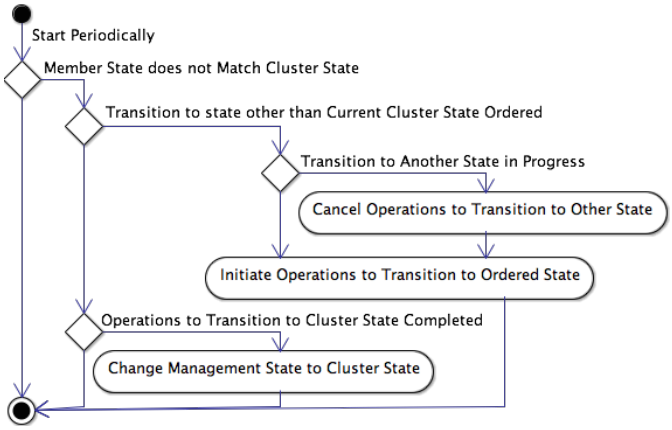

Fig. 5. Cluster instance State Management

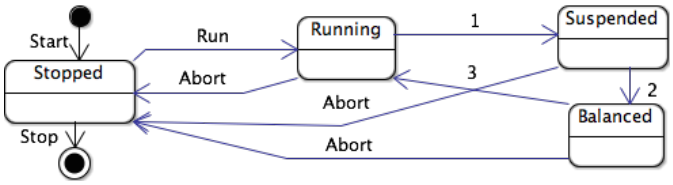

Fig. 6. Load Balancing State Transitions

instances have the initial state s_x. Therefore, if a cluster state transition occurs from $s_1 \rightarrow s_2$ and onto s_n, the cluster state will not transition to state s_n until all cluster instances have reached state s_2. In abnormal cases such as resetting to clear errors, the cluster state may transition to an initial state s_1, forcing all cluster instances to also reset.

The Cluster Master rules used to trigger load balancing may include virtualised node addition, modification, or removal notified over the NFV Os-Ma interface, addition or removal of application cluster instances by the OSS distribution mechanism, or an imbalance in the load being borne by cluster instances as reported by cluster instances in their metrics.

The Cluster Master may be selected in any manner that uniquely identifies it among its peer instances; such as by being the oldest instance of the cluster or selecting the cluster instance with the lowest unique identity. If this instance is shut down, another instance takes up the role of monitoring and updating the cluster state on its next periodic run.

B. Load Balancing State Transitions

The state transitions of a load balancing operation are shown in Fig.6. When the Cluster Manager decides to initiate load balancing, it sets the cluster state to *Suspended* (Transition 1) and waits for all instances to reach that state. Each instance observes the new cluster state and transitions to state *Suspended*. Typically, during this transition, an instance would then gracefully complete its ongoing management operations towards VNFs and close connections over the NBI interface to VNFs in an orderly manner.

When all cluster instances have reached state *Suspended*, the Cluster Manager sets the cluster state to *Balanced* (Transition 2). All cluster instances independently execute an identical asynchronous load balancing function, redistributing the management load across the instances. Cluster instances can execute the asynchronous function independently to determine what their load should be because the cluster state and metrics of all active instances is shared and available to all cluster instances.

When all cluster instances have reached state *Balanced*, it sets the cluster state to *Running* (Transition 3) and waits for all instances to reach that state. Each instance transitions to state *Running*. Typically, an instance would open connections over the NBI interface to VNFs and initiate management operations during this transition.

C. Asynchronous Load Balancing Function

The load balancing function to distribute VNF instances to a single management application instance must operate asyn-

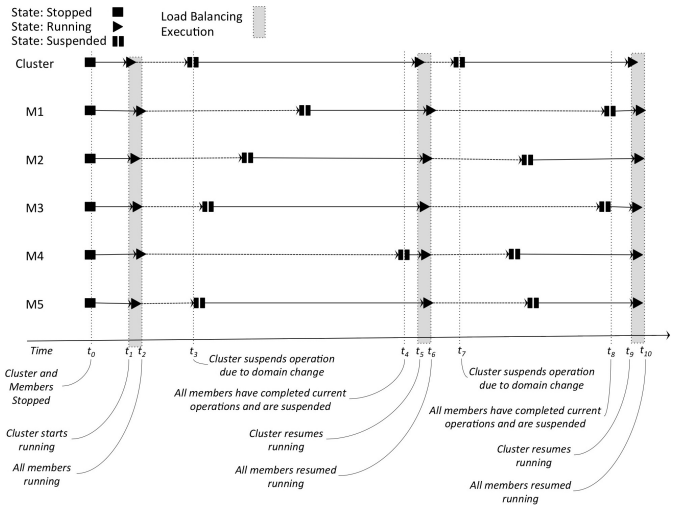

Fig. 7. Operation of the Algorithm in a Cluster with Six Instances

chronously and independent of other management application instances or their state.

Given V, a set of n VNF instances ($|V| = n; V_i \in V$) to be partitioned among a set management instances M of size m ($|M| = m; M_j \in M$), function $f(j, \ldots)$ must derive a subset of V for processing by M_j. L is a set of subsets of V: $(|L| = m; L_x \in L; L_y \in L; L_x \subset V; L_y \subset V; L_x \neq L_y \neq \emptyset; L_x \cap L_y = \emptyset; \{L_0 \cup \cdots \cup L_x \cup \cdots \cup L_m\} = V; L_x := f(x, V, PC, PI_x, MI_x))$.

Therefore, given function $f(\ldots)$, the subset of VNFs L_j to be processed by management application cluster instance M_j, can be derived asynchronously and independently.

IV. Operational Scenario

A typical operational scenario of the approach is shown in Fig.7. Here a management application is executing, and is using six autonomous instances, M_1 to M_6 to process the management load.

At time t_0, the cluster and all instances have state *stopped*. The application starts at time t_1.

The Cluster Master sets the cluster state to *running* at time t_1 (see Fig.4). The periodic operations of instances M_1 to M_6 read the cluster state change and trigger execution of their operations to transition from state *stopped* to state *running* (Fig.5). In this example, the load balancing function (See §III-C) is executed in each instance, which allocates a subset of the managed network elements to each instance. Following execution of the load balancing function, each instance establishes management sessions towards their allocated network elements and begins execution of management tasks towards those network elements and sets its state to *running*. By time t_2, all instances have state *running*.

From time t_2 to time t_3, the application executes as normal; no load balancing is performed. At time t_3, the Cluster Master detects that new network elements have been added to the network and sets the cluster state to *suspended*. The periodic operations of instances M_1 to M_6 read the cluster state change and trigger execution of their operations to transition from state *running* to state *suspended*. Each instance orders completion of management tasks and suspends initiation of any new tasks. Once all tasks have stopped, each instance sets its state to *suspended*. By time t_4, all instances have state *suspended*.

The Cluster Master again sets the cluster state to *running* at time t_5. The periodic operations of instances M_1 to M_6 once again read the cluster state change and trigger execution of their operations to transition from state *suspended* to state *running*. Load balancing is again executed and sessions towards network elements are established again. By time t_6, all instances have state *running*.

From time t_6 to time t_7, the application executes as normal; no load balancing is performed. At time t_7, The Cluster Master detects that the cluster load is unbalanced, and sets the cluster state to *suspended*. The periodic operations of instances M_1 to M_6 runs the operations described above to transition to state *suspended*. By time t_8, all instances have state *suspended*.

The Cluster Master again sets the cluster state to *running* at time t_9; by time t_{10}, all instances have again transitioned to state *running*.

V. Indicative Performance

The time to initiate or terminate a VNF is likely to be of the order of minutes. Any load balancing algorithm should execute in the order of tens of seconds so as not to significantly increase the time taken for a NFV to come into service.

We carried out measurements of the duration of load balancing operations in our lab to ensure that the performance of the algorithm is adequate for use in a NFV deployment and did not significantly add to the duration of a NFV start. We triggered load balancing by introducing groups of network elements and observed the time taken for the application itself and all instances to transition from state *running* back to state *running* (see Fig.6). We executed 10 runs for each network element group size, giving a total of 100 experimental runs.

For these measurements, we deployed 30 instances of the event loading application in 10 JVMs with each JVM running three instances of the application. We ran all the JVMs on a HP DL 380 G8 server with 32 CPUs and 250 GB of memory running Red Hat Enterprise Linux 6.3. Each instance collected event files using secure copy (*scp*) from simulated WCDMA and LTE network elements.

The measurement results shown with box plots in Fig.8 show that all load balancing operations execute in less than 80 seconds, the average load time was 41 seconds. The red line in Fig.8 fits the load balancing time median values as the number of network elements increase, and shows a slowly increasing trend from just over 30 seconds at 1000 NEs to 50 seconds at 10,000 NEs, a desirable characteristic for such an algorithm.

The large spread in load balancing times are caused by the implementation of the algorithm in the application. As explained in §III-B, when load balancing commences, each instance completes its ongoing operations. In measurement runs where large load balancing times are observed, the

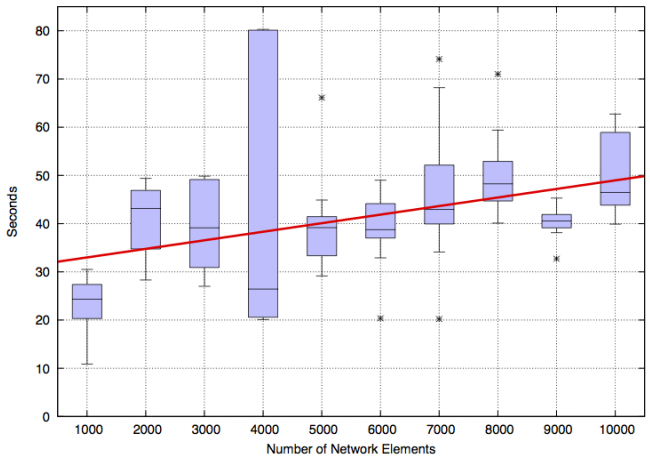

Fig. 8. Load Balancing Times: State *running* to State *running*

application instance was executing a batch of file copies from network elements, resulting in a longer time to transition from state *running* to state *suspended*. However, this clearly shows that ongoing operations gracefully complete without data loss and all connections were gracefully terminated. Fairer measurements of the algorithm's performance would measure the time taken to transition from state *suspended* to state *balanced* as such measurements eliminate the application specific aspects of load balancing.

VI. Implementation and Deployment

A proof of concept implementation of the algorithm has been developed and verified as an extension to the event processing module of the Ericsson Network IQ ENIQ) Events system [10]. The algorithm is used to balance the event collection load from nodes in mobile telecommunications networks, replacing the statically provisioned collection deployment described in [11].

The algorithm implementation has been deployed in two customer networks, one in the Americas and another in East Asia; Fig.9 shows a typical such deployment. Each instance periodically collects files containing events using secure copy (*scp*) from WCDMA and LTE network elements and a *mod()* function is used as the load balancing function (see §III-C). In both networks, prior to deployment of the algorithm, redistribution of the load across application instances required a restart of the collection system, which was performed in the maintenance window. Deployment of the algorithm allowed balancing of the file collection load across instances automatically; the need for system restarts was removed.

VII. Summary

The algorithm presented here solves the key problem of state aware load balancing of application instances controlling network elements running as or in VNFs. The algorithm is scalable in the management system and in NFV virtualisation because it runs across an arbitrary number of application instances and can load balance across any number of VNFs

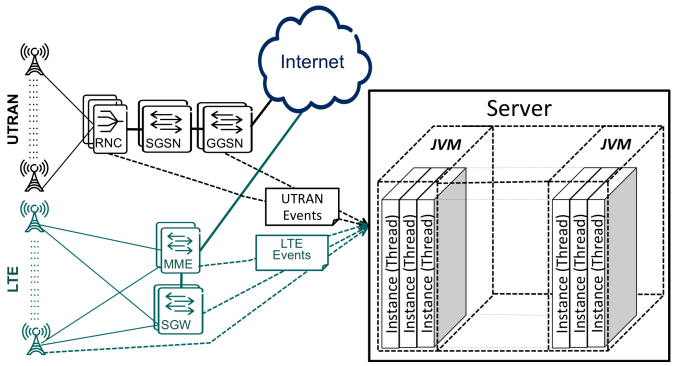

Fig. 9. Typical Deployment

deployed using NFV. Rules that specify the application load conditions that trigger the algorithm can be modified at run time. An asynchronous load balancing function executes independently in each application instance to balance the load. This allows the algorithm to balance the current application load fairly across all application instances and to allow application instances to spin up or spin down as application load increases and decreases.

References

[1] S. Landstrom, A. Furuskar, K. Johansson, L. Falconetti, and F. Kronestedt, "Heterogeneous Networks – Increasing Cellular Capacity," *Ericsson Review*, no. 2, pp. 34–38, 2011.

[2] ETSI, "Network Functions Virtualisation (NFV); Architectural Framework," ETSI, Tech. Rep. NFV 002, October 2013.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[4] J. A. Wickboldt, L. Z. Granville, F. Schneider, D. Dudkowski, and M. Brunner, "Rethinking Cloud Platforms: Network-aware Flexible Resource Allocation in IaaS Clouds," in *Integrated Network Management (IM), 2013 IFIP/IEEE International Symposium on*, May 2013.

[5] B. B. Nandi, A. Banerjee, S. C. Ghosh, and N. Banerjee, "Dynamic SLA Based Elastic Cloud Service Management: A SaaS Perspective," in *Integrated Network Management (IM), 2013 IFIP/IEEE International Symposium on*, May 2013.

[6] L. Richardson and S. Ruby, *RESTful Web Services*, 1st ed. O'Reilly Media, Inc., 2007.

[7] IETF, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks," RFC 3411, IETF, Tech. Rep. RFC 3411, Dec. 2002.

[8] ——, "NETCONF Configuration Protocol," RFC 4741 (Proposed Standard), IETF, Tech. Rep. RFC 4741, Dec. 2006.

[9] P. Gustas, P. Magnusson, J. Oom, and N. Storm, "Real-time Performance Monitoring and Optimization of Cellular Systems," *Ericsson Review*, no. 1, pp. 4–13, January 2002.

[10] Ericsson, *Ericsson Network IQ Events*, Ericsson, 2014.

[11] S. Achuthan and J. O'Meara, "A System for Monitoring Mobile Networks using Performance Management Events," in *Integrated Network Management, 2013. IM '13. IFIP/IEEE International Symposium on*. IM 2013, May 2013.