Extending Sensor Networks into the Cloud using Amazon Web Services

Kevin Lee[1], David Murray[1], Danny Hughes[2,3], Wouter Joosen[3]

[1]School of Information Technology, Murdoch University,
Murdoch 6150, Western Australia, Australia
{Kevin.Lee, D.Murray}@murdoch.edu.au

[2] Department of Computer Science and Software Engineering,
Xi'an Jiaotong-Liverpool University,
Suzhou Industrial Park, Suzhou, China,
Email: daniel.hughes@xjtlu.edu.cn

[3] Distrinet, Katholieke Universiteit Leuven,
3001 Leuven, Belgium
Email: wouter.joosen@cs.kuleuven.be

***Abstract*—Sensor networks provide a method of collecting environmental data for use in a variety of distributed applications. However, to date, limited support has been provided for the development of integrated environmental monitoring and modeling applications. Specifically, environmental dynamism makes it difficult to provide computational resources that are sufficient to deal with changing environmental conditions. This paper argues that the Cloud Computing model is a good fit with the dynamic computational requirements of environmental monitoring and modeling. We demonstrate that Amazon EC2 can meet the dynamic computational needs of environmental applications. We also demonstrate that EC2 can be integrated with existing sensor network technologies to offer an end-to-end environmental monitoring and modeling solution.**

I. Introduction

The use of Sensor Networks allow data from the physical world to be collected from an array of devices at different locations. This environmental data is collected and transmitted through a number of sensor nodes to a gateway and from there to the modelling back-end. Aggregation and pre-processing is often performed to reduce power consumption before data is transmitted to the application user. Applications which use data from Sensor Networks include environmental monitoring [1], medical computing [2] and industrial automation [3].

Sensor Networks consist of a number of nodes, often called motes, which cooperate to complete their task of collecting raw data and returning this to the application back-end. Motes generally consist of a small footprint embedded board with a low-power general purpose CPU, small amount of memory, small solid state storage and a limited power source. They include a variety of interconnects to external storage, network connectivity and of course the sensor(s) themselves. Motes may be programmed using a low-level programming language to do only a single task or may run a minimal Operating System such as TinyOS [4], which supports multiple tasks.

A key shortcoming in current sensor network approaches is a lack of elasticity in both sensing resources and computational resources.

- Despite recent advances, the cost of deploying, managing and maintaining sensor network infrastructure remains a key barrier to the use of Wireless Sensor Networks (WSN) technologies, especially for applications with a short life-cycle. Furthermore, as argued in [5], we believe that exposing sensing resources using a cloud-like model is a promising approach to promoting re-use in this domain.
- In terms of computational resources, environmental monitoring and modeling applications are typically tightly-coupled with their environment and have unpredictable computational demands. For example: during the extreme flooding in Europe in 2006, large scale Grid Computing resources had to be manually re-purposed to run flood modeling software and connected to live sensor data in order to provide timely flood predictions [6]. Computational resources have been similarly re-allocated on an emergency basis to support modeling of hurricanes and oil spills [7].

This paper proposes that the elastic computation model provided by the Cloud is well suited to meeting the unpredictable computational demands that are generated by environmental sensing and monitoring applications. Specifically, we demonstrate through evaluation that Cloud Computing resources are sufficiently elastic to deal with the unpredictable loads generated by real-world sensing applications. Furthermore, we

demonstrate that it is feasible to connect resource constrained sensor nodes directly to the extensible computational resources offered by the Cloud.

The remainder of this paper is structured as follows. Section II provides a background of the areas of Sensor Networks and Cloud Computing. Section III discusses the benefits of integrating the domains of Sensor Networks and Cloud Computing. Section IV presents a strategy for providing elastic computational capacity for environmental modelling applications using the Amazon Elastic Computing Cloud service. Section V describes a proposed architecture for an integrated environmental monitoring and modelling system. Finally, Section VI presents some conclusions and future work.

II. Background

This section provides an overview of the technologies relavent to this paper. Section II-A provide a background on Sensor Networks. Secton II-B provide a background on Cloud Computing and the Amazon Web Services suite of Cloud Computing services.

A. Sensor Networks

Wireless Sensor Networks (WSN) are composed of tiny computers knows as motes with embedded CPUs, low-cost sensors and low-power radios [8]. These motes form self-organizing networks that are capable of sensing and reacting to the physical world. To date, the WSN research community has primarily focused upon providing in-network support for sensor network application.

These research efforts have resulted in a variety of special-purpose Operating Systems for sensor nodes including TinyOS [4], Contiki [9], Squawk [10] and Lorien [11]. The research community has also been very active in the development of programming abstractions for wireless sensor networks include specialized component models such as NesC [12], OpenCom [13] and LooCI [14] along with macro-programming approaches such as TinyDB [15] and Kairos [16].

The research community has also been prolific in developing networking support for WSN at all levels of the network stack from the link-layer [17], [18] to application-level networking [1]. IP-based networking approaches include uIPv6 [19] and 6LowPAN [20] are particularly promising as they allow for seamless integration of sensing resources into distributed applications and the cloud.

A key shortcoming of current research efforts is a lack of consideration of the WSN back-end. Sensor networks rarely, if ever, operate in isolation and are much more commonly connected to back-end modeling infrastructure. Examples from real world sensor-network deployments include computational models of flooding [1], pollution [21] and hurricanes [7]. Currently, developers implement and connect to back-end computational facilities in an ad-hoc manner, with little development support available. The situation is further complicated by the necessity of elastic computation, which is required to deal with environmental dynamism in the face of emergency events. Currently, there is little or no support for modeling computational elasticity for sensing applications. Subsequently, emergency situations such as the April 2010 oil spill in the Gulf of Mexico necessitate manual re-tasking of computational facilities together with application refactoring [7].

This paper advocates the implementation of environmental sensor network models using the extensible computational resources that are available in the Cloud. It argues that sensor network applications should be able to directly extend or contract available modeling capacity provided in the cloud. The remainder of this paper demonstrates that current cloud facilities offer ample elasticity to deal with environmental modeling workloads and establishing a direct connection to these resources is feasible even in resource constrained WSN environments.

B. Cloud Computing

The concept of Cloud Computing has received a lot of attention in recent years [22]. Cloud Computing is fundamentally an evolution of distributed systems technologies such as Cluster Computing [23] and Grid Computing [24]. At its simplest, Cloud Computing is the provision of generic computing services over the Internet. There are many competing definitions of exactly what constitutes Cloud Computing [25], however, a broad consensus suggests that all Cloud Computing platforms include:

- Abstracted or virtualized resources.
- Elastic resource capacity.
- Programmable self-service interface.
- Usage-based pricing model.

Practically, Cloud Computing offers access to raw virtualized resources, high level support services and complete turnkey applications. Because of their wide scope and the public availability of Cloud Computing services; growth has been dramatic. There are many providers of Cloud Computing resources including: Google AppEngine [26], Rackspace Cloud [27], Slicehost [28] and Salesforce [29]

Amazon is a leading provider of flexible Cloud Computing resources with its Amazon Web Services (AWS) [30] suite of offerings, including the Elastic Computing Cloud (EC2), Simple Storage Service (S3) and higher level services such as SimpleDB and MapReduce. Amazon Web Services can be accessed through using web browser or client application tools. More directly they can be accessed using either the REST or SOAP protocols via the Amazon-supported APIs for Java, PHP, Ruby and .Net. Specifically:

- **EC2:** provides support for the dynamic instantiation and configuration of virtual machine instances.
- **S3:** provides support for creating and managing an extensible storage space for any kind of data.
- **SimpleDB:** provides support for setting up simple relational databases that allow developers to store and query data without needing to manage the database.
- **Elastic MapReduce:** provides support for performing data-intensive tasks with minimal setup and management overhead.

Together, the AWS suite of services provides rich support for the creation and management of elastic computation facilities. Section III will discuss how Cloud Computing resources can be leveraged in an elastic environmental monitoring and modeling scenario.

III. Integrating Sensor Networks into the Cloud

Our previous work argued that making WSN-based resources available via the Cloud to promote the integration of sensor data with Internet applications will reduced the cost of WSN infrastructure [5]. We refer to this vision as the Tangible Cloud, a brief overview of which is provided in Section III-A. This paper adds to our previous work by (i.) demonstrating Cloud Computing's capacity for supporting elastic sensing and modeling applications and (ii.) showing that it is feasible for sensor nodes to use and manage the proposed Cloud-based extensible modeling resources. The argument for this architecture is provided in Section III-B.

A. Exposing Sensor Resources in the Cloud

The provision of sensing resources in the Cloud extends the current domain of Cloud Computing to include the physical world. We refer to this vision as the Tangible Cloud [5]. As first class entities in the Cloud, sensor network devices can be used together with 3rd party Cloud resources. For example, the developer of an environmental monitoring and modeling application might compose sensing resources from the Tangible Cloud with storage and computational resources from the traditional Cloud. Section III-B explores how WSN resources can connected to the cloud.

B. Exploiting Cloud Resources in WSN Environments

As argued in the introduction, the research community has created a rich suite of in-network technologies including specialized programming paradigms [12], [14], network support [19], [17], Operating Systems [9], [4], [11] and middleware [31], [32]. However, development support ends at the network gateway, leaving the WSN developer to implement ad-hoc solutions in terms of modeling and analyzing data gathered from the WSN.

The implementation of back-end modeling resources can be particularly problematic due to the tight-coupling of sensing applications with their environment. Consider a flood modeling and warning application that makes use of live sensor data. During standard system operation, the timeliness of flood predictions may not be critical. However, when a flood occurs, the need for timely flood warnings increases. Using traditional computational technologies, extending computational capacity may require the manual allocation of additional resources and refactoring of the modeling application itself. In an elastic system such as the Cloud, it is possible to incrementally increase available computational power to meet application demands in a more fine-grained manner.

Section IV demonstrates how this elasticity can be realized using Amazons AWS Cloud resources and quantifies the degree of this elasticity. Section V then demonstrates that it is possible to directly connect sensor nodes to the proposed modeling resources.

IV. Elastic Computational Capabilities for Environment Modelling

A. Realising Computational Elasticity in Amazon EC2

This section discusses the use of Amazon EC2 for realising computational elasticity.

1) EC2 Architecture: Amazon's Elastic Compute Cloud (EC2) is, at its most basic, a web-service that allows its users to launch and manage Linux and Windows virtual machine instances hsoted on Amazon's computing resources. Virtual machines can be managed using a web-based console but also directly using SOAP or RESTful mechanisms which makes it possible to manage instances from any client that can issue HTTP/1.1 requests.

The process of launching instances on EC2 is the same using any of the management interfaces, as follows:

- An **Amazon Web Service account** must be created, enabling the user to pay for resources based on usage.
- An **Amazon Machine Image** (AMI) must be chosen. These are the operating system images that become instances when launched. Normal and micro instances are available at a variety of CPU and Memory configurations. There are a large amount available, including those prepared by Amazon, by third-parties and those that can be built by the user, for example, to host a specific model.
- **Key pairs** should be created that allow secure connections to be made to the instance.
- A **security group** should be defined that specifies what services can be accessed on the instances and who can access them.
- **Launching** of the instance takes place. Once the instance is setup a user can connect to it (e.g. using SSH) and configure it for the required application scenario.

The web-services API allows these tasks to be completed programmatically using standard HTTP calls with short XML descriptions or HTTP Queries. The most lightweight of these being HTTP Queries using the RESTful paradigm. For example, to list all available Ubuntu AMIs the HTTP Query would be the following:

```
http://ec2.amazonaws.com/
?Action=DescribeImages
&User.1=ubuntu
&...auth parameters...
```

This returns an XML document listing all the available Ubuntu AMIs along with specific details such as a description of the AMI, device mappings and kernel version. Instances can be started using simple a HTTP Query as follows:

```
http://ec2.amazonaws.com/
?Action=RunInstances
&ImageId=ami-e480aa90
&MaxCount=1
&MinCount=1
```

```
&KeyName=gsg-keypair
&Placement.AvailabilityZone=eu-west-1b
&...auth parameters...
```

Similarly, any operation involving the EC2 service can be controlled using either HTTP Query or SOAP. Therefore, the Amazon EC2 service has in-built support for lightweight configuration of Cloud resources using open standards.

2) Using EC2 to Support Dynamism: The Amazon EC2 service has built-in support for dealing with dynamic loads through the use of auto scaling. Auto scaling allows user-defined triggers to automatically control the launching and shutting down of instances. Users define *groups*, with associated triggers that will automaticaly scale EC2 resources based on bandwidth or CPU utilisation.

To provide the data for triggers, the Amazon CloudWatch service is configured to monitor the usage of the instance group. Cloudwatch measures the required metics such as bandwidth and/or CPU utilisation. A trigger is configured to respond to an increase in utiltisation through the use of two parameters:

- *Period* is the interval at which Amazon CloudWatch monitors the resource usage.
- *BreachDuration* is the amount of time the metric is required to be above the trigger point before starting another instance.

By adjusting the Period and BreachDuration parameters, the auto scaling can be used to respond rapidly to small changes in load or to only respond based on longterm trends in load changes. The actual values used will depend on the user's value of the performance improvement versus the cost increase associated with provisioning more resources. The following example defines a new autoscaling group with a maximum of 10 instances and sets the scaling algorithm to increase the number of instances when the CPU utilization reaches 50% for more than 300 seconds.

```
http://autoscaling.amazonaws.com/
?AutoScalingGroupName=sensorappn
&LaunchConfigurationName=e480aa90
&MinSize=0
&MaxSize=10
&Cooldown=0
&AvailabilityZones.member.1=eu-west-1b
&Action=CreateAutoScalingGroup

http://autoscaling.amazonaws.com/
?AutoScalingGroupName=sensorapn
?TriggerName=50percent
&MeasureName=CPUUtilization
&Statistic=Average
&Namespace=AWS/EC2
&Period=50
&LowerThreshold=0
&LowerBreachScaleIncrement=-1
&UpperThreshold=50
&UpperBreachScaleIncrement=1
&BreachDuration=300
&Action=CreateOrUpdateScalingTrigger
```

The EC2 Autoscaling service is AMI agnositic as it is only concerned with provisioning resources based on the load present on the autoscaling group. In order to use EC2 auto scaling successfully, the AMI and application must be designed to support parallel computation and instance replication.

As discussed in Section II-A, the environmental models which use sensor network-derived data are inherently parallel and would thus benefit from using cloud resources. As well as the application, the AMIs must also be configured to support load-balancing.

When increasing the number of instances for an application the current instance and state is not just replicated, instead, fresh AMIs are instantiated and data directed to them using the load-balancing algorithm. Therefore, to effectively support load-balancing, the AMI must be customised to support the application. This can be completed in two ways:

- Configure an AMI to contain the application setup ready to receive data.
- Configure the AMI to retrieve the application specification from Amazon's Simple Storage Service (S3) on instantiation and setup the application.

Using both these methods, the Amazon AutoScaling service will be able to balance the load across multiple instances; increasing and descreasing the number of instances as necessary. The auto scaling services can then direct data across the multiple instances. The method used depends on the flexiblity needed by the application.

B. Quantifying Elasticity of Amazon EC2

To evaluate if Amazon EC2 can effectively load-balance when subjected to dynamic loads; the following experiment was performed.

1) Experiment Setup: A single AMI was prepared using Ubuntu 10.04 as the base Operating System and a default install of Apache 2.2.14. This AMI was stored on Amazon Simple Storage Service (S3) ready to be instantiated by EC2. A single instance of this AMI was instantiated on a standard machine type *m1.large* which has 4 64-bit EC2 Computer Units and 7.5GB of memory. No location preference was given for this or any of the EC2 instances used to ensure the availability of resources. Amazon CloudWatch was turned on to enable the collection of statistics from the instance.

An Amazon AutoScaling group was created and configured, with load-balancing enabled, to monitor instance and scale depending on CPU utilization. The low CPU utilization threshold was set to 25% and the high utilization to 40%. The AutoScaling service was configured to increment or decrement the number of EC2 instances when these values are breached for more than two minutes.

To evaluate how Amazon EC2 AutoScaling responded to dynamic load, two further EC2 instances were used to provide load. Each was of standard type m1.xlarge, which has 8 64-bit

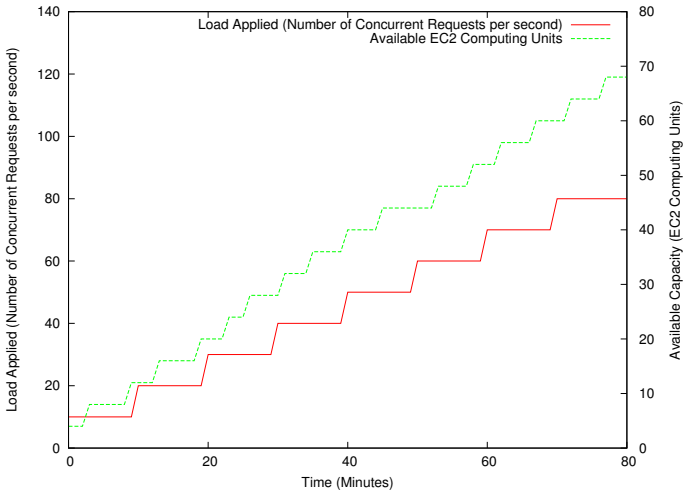

Fig. 1. Number of Available Amazon Compute Units with Increasing Load Applied being Applied

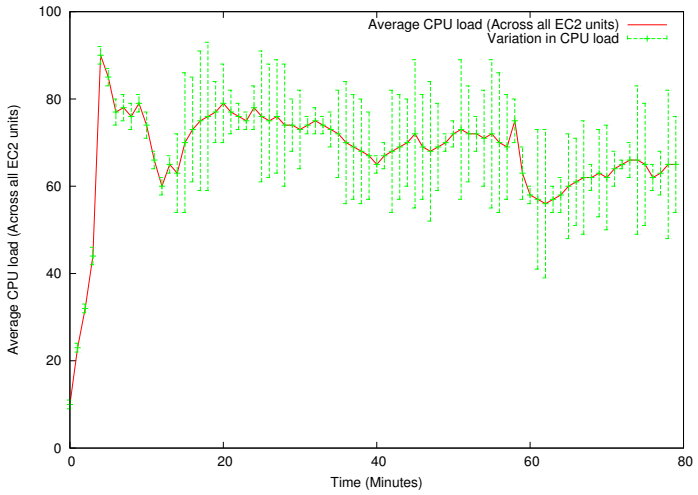

Fig. 2. Average CPU Usage on all amazon compute units over time

EC2 Computer Units and 15GB of memory. To provide load, the HTTP load testing and benchmarking tool Siege (version 2.69) was used to provide a constant measured load. Siege can be configured to perform a set amount of concurrent requests to a server on a target machine. For this experiment, applied load was simulated by using HTTP to request a small amount of server-side computation. These loads were directed at the DNS of the load-balancer, which forwarded the requests to selected instances.

2) Experiment Results: For this experiment the load applied is started at the level of 10 concurrent requests per second, then increased every 10 minutes over 80 minutes. The level of the load applied, the actual CPU usage on each instance, the number of instances available and transaction statistics were recorded.

Figure 1 illustrates the number of Amazon compute units that are available at any point during the experiment. It shows that as load was increased (left y-axis), the number of instances made available (right y-axis) by the Auto Scaling service also increased to handle the load. Using the Amazon Auto Scaling support, the instances appear to start almost instantly after the scaling trigger fires. The results show that, after the initial excessive load was applied there was always enough capacity available for the load applied. Furthermore, the increase in resource availability closely matches that of the applied load.

Figure 2 illustrates the average CPU usage on all virtual machines as the load was increased. It shows that, after the initial increase in load, the auto scaling and load balancing services successfully maintain balanced CPU usage across all available instances. The variance bars in Figure 2 show the fluctuating load between instances. The load balancer periodically increases and decreases load to different compute units.

To provide context, repeating this experiment with only a single instance results in a CPU utilization of 100% within 3 minutes. This CPU utilization was maintained for the length of the experiment whilst dropping an increasing number of connections. Conversely, providing enough CPU-capacity to meet the peak demand thoughout the experiment would have wasted energy and financial resources.

The Amazon load-balancing algorithm appears to unevenly deliver load to instances. Instead of load being evenly distributed, resulting in similar CPU usage on each note, load was sometimes unevenly distributed, resulting in sporadic CPU usage. This is illustrated in the variance of load in Figure 2. It is also observed that the load-balancer performs load-balancing based on source IP rather than per-connection. This must be considered when designing the use of EC2 resources into a large application scenario.

3) Summary: This experiment illustrates that Cloud Computing services can be used to provide computation to suppport dynamic load in scenarios such as those used to process data collected by sensor networks. Furthermore, obtaining, configuring and utilising these resources is simple, straightforward and relatively cheap. Using standard web-service technologies and open pricing makes this a viable for accessing computation resources.

V. Towards and Integrated Environmental Monitoring and Modelling System

This paper has made the case for the integration of sensor networks with the Cloud. We have shown, through experimentation, that existing Cloud Computing services such as Amazon's EC2 are sufficiently elastic to support the dynamic demands of real-time environmental monitoring and modeling solutions. This section shows how these features can be exploited using a prototypical implementation built using the LooCI WSN component model [14] and Amazon EC2 [30].

Figure 3 illustrates an overview of the architecture of our prototype. Components of the system are bound together through bindings of service provision (interfaces) and service requirement (receptacles). Interfaces are denoted by lines ending in balls, and receptacles are denoted by lines ending in

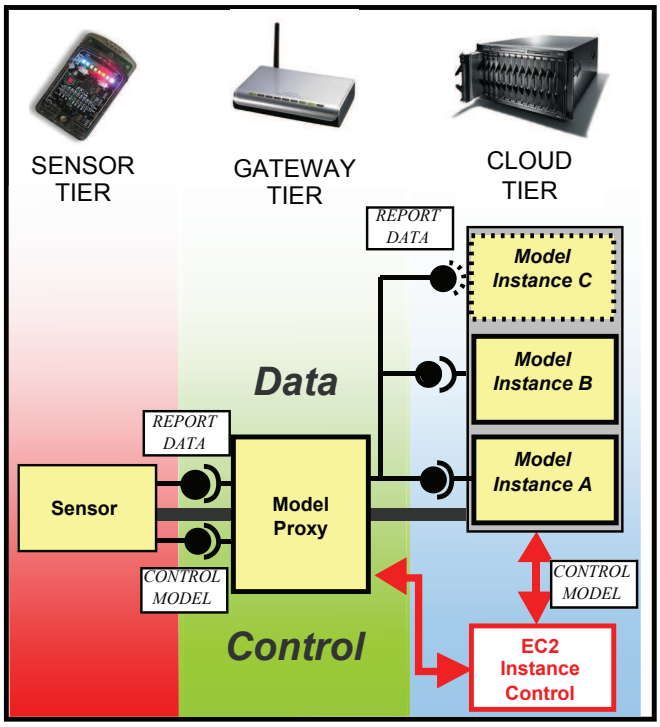

Fig. 3. System Overview

cups. Component bindings are shown as relationships between the interfaces and receptacles and are realized practically through the LooCI Event Bus [14]. SOAP bindings are shown as relationships between the EC2 instance control, model instances and model proxy. This implementation has three key tiers of functionality; Section V-A describes the Sensor Tier, Section V-B describes the Gateway Tier and Section V-C describes the Cloud Tier architecture. Finally, Section V-D introduces the API for the model.

A. The Sensor Tier Architecture

At the Sensor Tier, environmental data is gathered on low-power sensor nodes running the LooCI [14] middleware. Sensing functionality is realized using generic LooCI components which are bound over a loosely-coupled event-bus. Both the data and control plane are accessible in-network over the event bus as the model proxy offers LooCI interfaces to control the extensible model and report data. This eleminates the need to provide in-network SOAP/REST adapters and thus lowers overhead.

B. The Gateway Tier Architecture

At the Gateway Tier, the Model Proxy acts as a bridge, allowing LooCI sensing components to parameterize the extensible model over the event bus via SOAP-based EC2 functionality. Additionally, the model proxy relays environmental sensor data from the sensor network to the most optimal model instance. Model selection may be based upon a range of load balancing techniques, however, this is outside of the scope of this paper. This process is transparent to the sensing components and the application developer allowing for simple elasticity in computation.

C. The Cloud Tier Architecture

In the Cloud Tier, an extensible model is realized as an Auto Scaled set of EC2 instances. Each model is implemented in LooCI and therefore may connect directly to the model proxy over the event bus. Model functionality is encoded in a pre-configured EC2 AMI image as described in the previous section and when the existing pool of instances approaches a CPU utilization trigger, a new instance may be dynamically instantiated. Once it is instantiated, a reference is passed to the model proxy, which includes the instance in the Model pool, providing new capacity. Capacity may also be added manually.

D. API for the Model Proxy

The model proxy exposes a standard Java API for gateway applications and a distributed version of this same API over the in-network LooCI event bus. It is infeasible to reproduce the full API here, thus we focus upon the model control and data relay methods.

Model Control Methods:

- *modelRef startExtensibleModel()*
  This method starts a new extensible model, which begins with a single model instance, returning a unique modelRef (the IP address of the first instance) which is used to identify this model.
- *stopExtensibleModel(modelRef)*
  This method stops all model instances belonging to the unique extensible model identified by the parameter modelRef.
- *setExpandThreshold(cpuLevel)*
  This method sets the CPU utilization at which another model instance should be added to the extensible model.
- *setContractThreshold(cpuLevel)*
  This method sets the CPU utilization at which redundant model instances should be removed from the extensible model.
- *setBreachDuration(duration)*
  This method sets the breach duration the CPU utilization must be above ot below before the instances should be added or removed.
- *incrementCapacity(modelRef)*
  This method manually expands the specified extensible model with a single additional EC2 instance.
- *decreaseCapacity(modelRef)*
  This method manually contracts the specified extensible model by removing a single additional EC2 instance.

Data Relay Methods:

- *report(sensorData, modelRef)*
  This method sends the specified sensor data to one of the instances that comprises the extensible model identified by modelRef. A range of load balancing algorithms may be applied to ensure a uniform load is generated.

VI. Conclusions and Future Work

This paper has argued for the integration of sensor networks and Cloud Computing to support the dynamic loads that are generated by environmental applications. It has demonstrated how sensor networks can be combined with Cloud Computing to allow the offloading of resource-intensive tasks to the Cloud. An experiment was performed to show that the elastic nature of Amazon EC2 can support the dynamic loads provided by these applications. We have also provided an architecture that shows how existing sensor network technologies can be integrated with Cloud Computing.

Our future work will focus on building a proof-of-concept application that uses extensible Cloud-based modeling resources to provide reliable identification of suspect vehicles in variable traffic situations. In this scenario, the Cloud will be used to host image analysis models that are capable of identifying suspect vehicles that police officers are searching for. As traffic increases, more models will be launched to provide a mechanism that can reliably identify suspect vehicles even in high volumes of traffic.

Acknowledgments

We acknowledge the support of Amazon Web Services for providing Cloud Computing resources for this research under their AWS in Education grant scheme.

References

[1] Hughes D., Greenwood P., Coulson G., Blair G., Pappenberger F., Smith P., and Beven K. An experiment with reflective middleware to support grid-based flood monitoring. In *in Wiley Inter-Science Journal on Concurrency and Computation: Practice and Experience, vol. 20, no 11, November 2007, pp 1303-1316*, 2007.

[2] Stankovic J. A., Cao Q., Doan T., Fang L., He Z., Kiran R., Lin S., Son S., Stoleru R., and Wood A. Wireless sensor networks for in-home healthcare: Potential and challenges. In *in proc. of Workshop on High Confidence Medical Devices Software and Systems (HCMDSS)*, 2005.

[3] Pohl A., Krumm H., Holland F., Stewing F. J., and Lueck I. Service-orientation and flexible service binding in distributed automation and control systems, 2008.

[4] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. Tinyos: An operating system for sensor networks. In *in Ambient Intelligence*. Springer Verlag, 2004.

[5] K Lee and D. Hughes. System architecture directions for tangible cloud computing. In *International Workshop on Information Security and Applications (IWISA 2010), in Qinhuangdao, China, October 22-25*, 2010.

[6] Paul Smith, Danny Hughes, Keith J. Beven, Philip Cross, Wlodek Tych, Geoff Coulson, and Gordon Blair. Towards the provision of site specific flood warnings using wireless sensor networks. In *Meteorological Applications, Special Issue: Flood Forecasting and Warning, volume 16, number 1*, pages 57–64, 2009.

[7] Faith Singer-Villalobos. Scientists produce 3-d models of bp oil spill in gulf of mexico using ranger supercomputer, univeristy of texas at austin press release, available at http://www.utexas.edu/news/2010/06/03/tacc_ranger_oil_spill/. June 2010.

[8] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM.

[9] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, November 2004.

[10] Doug Simon, John Daniels, Cristina Cifuentes, Dave Cleal, and Derek White. The squawk java virtual machine, sun microsystems, 2005.

[11] Barry Porter and Geoff Coulson. Lorien: a pure dynamic component-based operating system for wireless sensor networks. In *MidSens '09: Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*, pages 7–12, New York, NY, USA, 2009. ACM.

[12] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, New York, NY, USA, 2003. ACM.

[13] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan. A generic component model for building systems software. In *ACM Transactions on Computer Systems, pp 1-42, Vol. 26, No. 1*, 2008.

[14] D. Hughes, K. Thoelen, W. Horre, N. Matthys, S. Michiels, C. Huygens, and W. Joosen. Looci: A loosely-coupled component infrastructure for networked embedded systems. In *in the proceedings of the 7th International Conference on Advances in Mobile Computing & Multimedia (MoMM09), December*, 2008.

[15] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[16] Ramakrishna Gummadi, Nupur Kothari, Ramesh Govindan, and Todd Millstein. Kairos: a macro-programming system for wireless sensor networks. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 1–2, New York, NY, USA, 2005. ACM.

[17] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor asystems*, pages 95–107, New York, NY, USA, 2004. ACM.

[18] Chunsheng Zhu, Yuanfang Chen, Lei Wang, Lei Shu, and Yan Zhang. Smac-based proportional fairness backoff scheme in wireless sensor networks. In *IWCMC '10: Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, pages 138–142, New York, NY, USA, 2010. ACM.

[19] Dogan Yazar and Adam Dunkels. Efficient Application Integration in IP-based Sensor Networks. In *Proceedings of ACM BuildSys 2009, the First ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings*, Berkeley, CA, USA, November 2009.

[20] Jonathan W. Hui and David E. Culler. Extending ip to low-power, wireless personal area networks. *Internet Computing, IEEE*, 12(4):37–45, July-Aug. 2008.

[21] Wataru Tsujita, Akihito Yoshino, Hiroshi Ishida, and Toyosaka Moriizumi. Gas sensor network for air-pollution monitoring. *Sensors and Actuators B: Chemical*, 110(2):304 – 311, 2005.

[22] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, 2009.

[23] Rajkumar Buyya. *High performance cluster computing / edited by Rajkumar Buyya*. Prentice Hall PTR, Upper Saddle River, N.J. :, 1999.

[24] I. Foster and C. Kesselman. Concepts and architecture. In I. Foster and C. Kesselman, editors, *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kauffman, San Francisco, second edition, 2004.

[25] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009. 1496100.

[26] Eugene Ciurana. *Developing with Google App Engine*. Apress, Berkely, CA, USA, 2009.

[27] Rackspace. the rackspace cloud. http://www.rackspacecloud.com, 2010.

[28] Slicehost. Slicehost cps hosting. http://www.slicehost.com, 2010.

[29] Salesforce. Salesforce crm. http://www.salesforce.com, 2010.

[30] Amazon. Amazon web services. http://aws. amazon.com, 2010.

[31] Christophe Huygens, Danny Hughes, Bert Lagaisse, and Wouter Joosen. Streamlining development for networked embedded systems using multiple paradigms. *IEEE Software*, 27:45–52, 2010.

[32] S. Hadim and N. Mohamed. Middleware: middleware challenges and approaches for wireless sensor networks. *Distributed Systems Online, IEEE*, 7(3):1 –1, mar. 2006.