Dynamic Data Storage Estimation for Multiple Concurrent Applications Using Probability Distribution Modeling in WSNs

Mo Haghighi

Abstract—**Wireless sensor networks (WSN) have become a mainstream technology for environmental monitoring and observing various variables of interest over extended periods of time via large-scale networks of sensors. WSNs have a wide range of applications including wildfire detection, healthcare, military, and habitat monitoring. In all such application areas, gathering and then relaying captured data to a central unit is often considered the primary task of the network. Scientific analysis however often requires WSNs to capture and store variables for long periods of time. Storing and managing flows of data tend to be challenging issues because WSNs often consist of nodes with limited processing, memory, and power resources. Therefore the software layer in WSNs needs to implement an efficient data storage allocation mechanism in order to provide sufficient memory space for multiple applications. In this paper we propose a novel statistical approach for estimating applications storage requirements. Our proposed mechanism has been originally developed and implemented in a new WSN middleware called Sensomax, which is an agent-based decentralized middleware with multiple concurrent applications support for dynamic data gathering in WSNs. The mechanism described here proved to be an effective technique for proactively allocating memory to multiple applications with different operational paradigms.**

Index Terms—**Storage, WSN, probability, distribution, concurrency.**

I. Introduction

Devising storage allocation mechanisms has always been a challenging task for application developers for both conventional systems such as PCs, and for a wide range of embedded devices including wireless sensor networks (WSNs). The most challenging issue often comes in allocating appropriate storage space to different applications based on their current requirement and on how their storage requirement may change over time. In conventional systems such issues are not as problematic as in the embedded world, mainly because in conventional systems storage-space is usually not scarce; on the other hand, embedded devices such as WSNs suffer from extreme scarcity of hardware resources most notably their limited memory and energy resources. This issue of limited resources in WSNs is one of the primary factors that influences programming and deployment of these devices. WSN applications often fall into one of two main categories: either real-time online monitoring, or long-time offline observation. In the latter case, the network is required to observe variables of interest over an extended period of time, storing the captured data for subsequent relaying to a central unit. The captured data may be stored in a raw (or lossless-compressed) form or in some internally-aggregated form such as lossy-compressed or summary statistics. This requires WSN nodes to not only provide reasonable energy longevity for the application, but also to provide enough storage to retain sufficient data for the lifetime of the application. Recent research on WSNs has mostly focused on conserving energy to increase the life of WSNs, yet lack of sufficient storage could make the network as non-operational as a lack of energy does. WSN applications are usually required to operate unattended and in many cases need to be deployed in hostile environments or places where access to nodes is severely constrained, or impossible. Therefore efficient mechanisms with a fair extent of autonomy are a key requirement for WSNs' operating systems or middleware. Conventional WSNs often consist of tightly coupled hardware and software whereby very application-specific services are coded into the entire software stack running on each node. With recent advances in microelectronics and embedded systems design however, new WSN devices have been manufactured that can offer richer hardware resources and hence pave the way for dynamic general-purpose software to be accommodated in such devices.

Middleware is a software layer that lies between the application layer and hardware resources, managing communication, processing, storage and resource allocation amongst hardware resources and applications. Running an autonomous middleware such as Sensomax [1], [2], often requires resources that are capable of multi-tasking and running algorithms for aggregation and decision-making. Probably the most widely used hardware resources in WSN research are the *Mica* family [3] such as the *Micaz* and *Mica2* motes that are very often used with *TinyOS* [4], which is an operating system originally developed for Mica nodes. When it comes to memory and processor, Mica motes are extremely resource constrained. For example, the Mica2 hardware includes an Atmel processor operating at 16MHz and its memory includes 4Kb of RAM and 128Kb of flash storage. With such scarce resources, the tasks of developing autonomous software and storing large volume of data are very constrained. New devices such as the *SunSpot* [5] however provide a resource rich platform in a high-level development environment such as Java. SunSpot devices include a 400MHz ARM processor, 1Mb of RAM and 8Mb of data storage. The hardware features of SunSpot and the programming power of Java motivated us to design and

Manuscript received March 25, 2013; revised May 19, 2013.

Mo Haghighi is with the Department of Computer Science, University of Bristol, Bristol, BS8 1UB, UK and Large-Scale Complex IT Systems (LSCITS) (e-mail: Mo.Haghighi@bristol.ac.uk).

implement a novel dynamic middleware called *Sensomax* which runs on SunSpot nodes and also on *Raspberry Pi* single-board computers [6]. Sensomax incorporates a number of conventional systems' capabilities such as serving concurrent multiple applications, agent-based communication, dynamic runtime reconfiguration and decentralized execution of multiple operational paradigms. The detailed architecture of Sensomax has been described in previous publications [1] and [2], and is intended for future open-source release, but for completeness we provide a brief description of its high-level component interactions in the remainder of this section.

Sensomax hosts multiple applications concurrently and this feature requires many careful considerations in terms of fair allocation of resources amongst applications while maintaining a reasonable lifetime and meeting each application's performance requirements. This requires a number of trade-offs amongst different aspects of the network, which necessitate a number of autonomous decision-making processes at both node and network levels. Sensomax exploits autonomous controllers in several areas including the storage allocation and inter-cluster distributed processing. For the purpose of this paper, we focus on how Sensomax implements autonomous storage management and allocation, both globally and locally, to meet multiple application demands for different operational paradigms.

In WSNs, data storage can be categorized into two major sub-categories of *Local Storage* and *Collaborative Storage* [7]. In the former, data are stored into the sensing node; and in the latter, sensed data get relayed to other nodes for further processing or storage, such as the cluster-head or the base station. However, [8] as one of the earliest research attempts at WSN storage, proposes a context-aware storage mechanism using Geographical Hash Table (GHT) and categorises data storage into three fields of Local Storage (LS), External Storage (ES) and Data Centric Storage (DCS), in which, LS represents the same definition as the local storage in [7], ES and DCS on the other hand, can be described as sub-categories of collaborative storage. In ES, data get transferred to a central sink node without prior aggregation, whereas in DCS, data get relayed to a node with close proximity to the captured event. Several other researchers in WSN storage have tried to enhance the DCS with new techniques in order to improve the energy consumption model and data-gathering, such as [9].

Scientific applications often require data gathering on specific variables over extended periods of time in order to generate large datasets, sufficient for deriving reliable statistics. Therefore WSN middleware needs to provide a fair amount of storage capacity. When dealing with a single application, this issue can be much simpler as the entire storage becomes available to the application. The issue in that case is how to distribute data amongst nodes in order to take full advantage of storage available in all nodes. That often requires capturing data by one node and then relaying the data to other nodes for storage. However in Sensomax we are dealing with multiple applications running on a single node or on a cluster of nodes. This imposes a new challenge: how to allocate storage locally to multiple applications as well as distributing storage globally amongst other clusters and nodes. While many techniques have been proposed to deal with storage of a single WSN application, handling multiple concurrent applications requirements is something that has been overlooked in the WSN research literature.

Handling multiple concurrent applications often requires the middleware to have full knowledge of the amount of storage needed by each application. In principle, such issues can be solved by dividing the available storage fairly amongst all, However, WSN applications in dynamic environments have the tendency of changing their requirements due to changes in the external environment that is being sensed/monitored, alterations in application demands, or as a result of analysing the captured data. Dynamic applications constantly change their requirements and the middleware should be capable of dealing with those changes and actively reassessing the allocations. We believe applications should be able not only to actively reconfigure the network but also to implement the reconfigurations proactively in order to predict the application demands ahead of time. Such strategies require a great deal of autonomy and subsequently increase the amount of processing which leads to a shorter WSN lifetime. Therefore they should be implemented considering all aspects of the WSNs as well as taking the application requirements into account without imposing huge resource overheads.

In this document we propose a mechanism that gradually adapts to application storage demands and proactively allocates the available storage in an efficient manner. This mechanism requires the applications to run for a short while before training their captured data as a sample using statistical analysis. We propose a mechanism that here is described in terms of using standard statistical probability distributions such as the binomial, Poisson and Gaussian distributions which can readily be extended to other distributions. Our model uses these probabilistic techniques to predict how much storage is needed for each application and constantly reassess its strategy as more data become available. Based on our experiments, the more data are captured by the node, the more accurate the probabilistic models become. Each probabilistic model is represented by an algorithm, which is integrated in a standalone component and is executed by the middleware. It is worth noting that this process does not micro-model the data but it focuses on the frequency of event occurrences and how the overall patterns of data change over time. In the next section we will look at some of the prior literature on statistical analysis in WSNs as well as existing WSN storage techniques

II. Architecture

Our proposed Sensomax storage protocol works in a step-by-step allocation fashion, whereby, on the node level, each node's available storage space is assessed multiple times during its lifetime. Every assessment may increase or decrease the node's storage based on the number of applications running on the node and their runtime requirement changes. On the application level, every application's available storage is also subject to change, based on the total allocated storage to the current running applications and number of new concurrent applications to be added to the node. The main objectives of our proposed architecture can be summarized as follows:

1) Decentralized data storage estimation for each application locally by each node.(Local Storage)
2) Exploiting popular statistical data analysis that are easy to implement using Java and do not impose high footprint on resources.
3) Facilitating more efficient data storage estimation for cluster-heads by breaking down the job among the members. (Cooperative Storage)
4) Revalidating the allocated storage periodically to meet the application demands dynamically.

As we explained earlier, Sensomax can host multiple applications hence the logical choice for allocating storage space to each application can be initially implemented simply as the equal division of the available storage. Such a simple equation could only work if the node receives all applications at the same time, if all applications have equal storage requirements, and if the requirements remain constant over time. However, such a scenario only applies to highly customized and application-specific WSNs with a fixed number of applications, that seem unsuitable and highly inefficient for modern WSN applications and next-generation WSN node hardware. Hence, we are dealing with multiple concurrent applications that are added, removed and modified dynamically at runtime on node, cluster and network levels. This necessitates a dynamic allocating process that satisfies all these characteristics whilst managing storage efficiently.

A number of methods, including those mentioned in the previous section, were considered and we have synthesised their best practices, combined and extended them, into a uniform mechanism in Sensomax that overcomes several memory allocation complexities. In order to dynamically modify the storage allocations, applications' storage requirements not only need to be calculated actively but also proactively. By this we mean that the middleware needs to respond to their storage demands not only when more storage is required but also be able to predict their required storage ahead of time. Such predictions can be achieved through appropriate probabilistic modelling.

To clarify our reasoning for developing probabilistic models of future resource needs and demands, further explanation of the problem needs to be provided here. WSN applications can be classified into four categories of Time-driven, Event-driven, Data-Driven and Query-driven. As we explained earlier, Sensomax handles each application category exclusively. Therefore applications of each type are executed in complete isolation and its requirements are also processed in a context-aware environment. In the same way, event-driven applications monitor specific variables over a period of time and only store the values if the event of interest meets some criteria e.g. typically when the variable exceeds over/falls below a certain threshold. For this type of application, there exist some uncertainties concerning when and how often such events happen. Obviously predicting exact occurrences of those events is impossible, but this is where probabilistic modelling techniques can help us to train event-driven data for a short period in order to estimate the frequency of event occurrences and hence its required storage.

In this section we propose prediction mechanisms using techniques based on discrete and continuous probability distributions, facilitating the temporal modelling of key variables in an event-driven manner that run for a finite period of time.

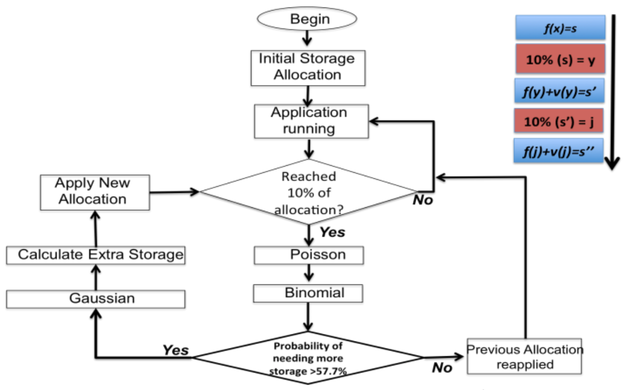

Fig. 1. Continuous storage estimation every 10% of the previously allocated storage or time.

In order to initiate these probabilistic models, some approximate preliminary data about the observed variables are required, such as how frequently a variable may change over a certain period of time. These preliminary data may not be extracted from the application itself unless the application runs for a short "bootstrap" or "ramp-up" period. Sensomax consists of a step-by-step technique in which the application runs for 10% of its total observation period or 10% of its initial allocated storage space in order to obtain sufficient data on its behavioural pattern. As Fig. 1 shows, every incoming event-driven application is assigned with an arbitrary storage. The amount allocated is for data training purposes only and purely depends on the remaining storage and the number of applications running in the node. The application may start capturing and continues until either it has filled 10% of its allowed space, or it has run for 10% of its operational period (for cases that have specific starting and ending times,) or whichever comes first. After passing its 10% testing period, the captured data are then input for the probabilistic modelling. The results from the modelling then determine an estimated storage space for the remaining lifetime of the application. This process continues for every 10% of the allocated storage until after a stable storage is determined.

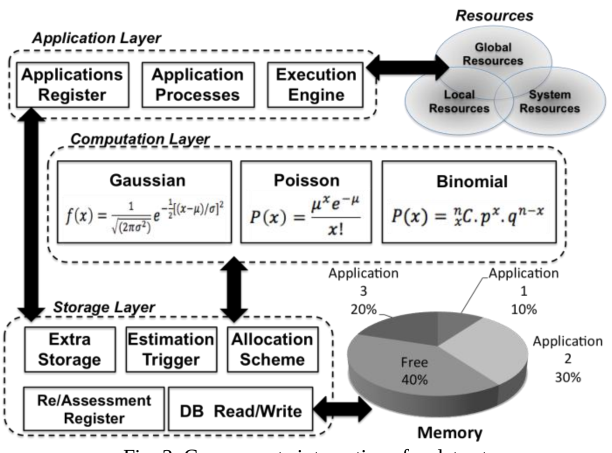

Fig. 2. Components interactions for data storage

Our proposed middleware follows a components-based design where each component represents a software module in charge of a set of homogenous sub-tasks aimed at fulfilling a higher-level task. Such modulated design helps the application developers to envision the entire system as a collection of interactions amongst various modules and in turn facilitates the development process by interpreting the application requirements into a set of coordinated interactions amongst modules.

As Fig. 2 shows, the process of applying multiple algorithms (from the computation layer) on the captured can be seamlessly programmed into the system by exploiting the modules' functionalities in each layer. Different applications can access the memory directly while the allocation process is done inside the storage layer.

In order to use a binomial model, events should be described in terms of success and failure over a fixed number of trials. Therefore this distribution is suitable for event-driven applications. However there requires a number of trials, whereby Sensomax translates the testing duration as the number of trials based on the events frequency pattern e.g. the trial number for two events happening in a second with 3 milliseconds interval is 1000 whereas the trial for two events happening in 60 seconds with 3 seconds interval is 60.

Assigning the number of trials for binomial distribution, in the way we have done, could be problematic in terms of accuracy when dealing with few number of events over a long period of time or a large number of events over a very short period of time. Therefore Poisson distribution is used as the primary data modelling to estimate the storage. One of the features of Poisson distribution is treating events independently, which is very much applicable to event-driven scenarios in WSNs, provided a single variable is captured by every instance of an application. Existing Poisson-based approaches in WSNs include: [10] and [11] approximate traffic flow in multi-sink sensor network via a Poisson process, [10] estimates the number of events occurring in the network and the number of corresponding packets to be generated and sent around the network via a model based on the Poisson distributions and [12] uses a Poisson model to determine the optimal number of cluster-heads in order to reduce the overall energy consumption of the network.

As Fig. 1 shows, captured data are trained using the Poisson distribution and then double-checked using the Binomial distribution. Based on our experiments, when the probability proved to be higher than 0.575 (57.5%) then the allocation needs to be applied (This will be shown in the evaluation section). Basically a number of different probability values (in percentage) were tested against the actual storage required (in percentage), and 57.5% proved to be a reliable confidence threshold. Once the application receives its allocated storage, it resumes its observation. After 10% of the allocated storage is used, the middleware assesses its storage based on the new captured data and its previous allocation. A new storage allocation is made, which could be more or less than the previous allocation. This process continues until a fair number of allocations are done, in which case the 10% checking interval could rise in order to reduce the number of assessments and save energy.

Based on our practical experiences, and many experiments, it is clear that some applications may dynamically change their observation requirements dramatically. In such cases the allocated storage will not be enough and the application will run out of storage very quickly before the next reassessment period is reached. Based on the aforementioned nature of event-driven applications, there requires an extra temporary space for cases where the application requirement overflows the allocated storage. The standard deviation of normal distribution is an appropriate measure to be added to the allocated space. The allocated probability using the Poisson and Binomial distributions can represent the mean of the normal distribution and standard deviation derived from the Gaussian distribution can represent the spread of probability around the mean value. Therefore we use the standard deviation to calculate the extra storage to prevent an application from running out of space. Standard variation is the square root of variance. In Fig. 1 and in the next section we have used variance instead of standard deviation just to conform to the common practice of statistical analysis. In a similar approach, [13] investigates Poisson- and Gaussian-distribution models for locating randomly deployed nodes and estimating the noise ratio, respectively. The distributional models are used to fuse the total number of detections based on their estimated locations in the network, to increase the accuracy of target detection.

III. Evaluation

We have conducted a number of experiments to prove the efficiency of using probability distributions for estimating the applications' required storage, on both cluster and node levels, using Sensomax running on a small WSN of 14 Sun Spot devices as prototypes, and also a bespoke simulator called SXCS which we wrote as a simulator/emulator designed for Sensomax.

For the first experiment, a virtual environment with 500 virtual nodes, running lightweight applications, which were interested in only 2 variables, was simulated for a period of 54 seconds. Each node was set to have 50 events on average, whilst running 5 concurrent applications. The objective of this experiment was to find the difference between how much storage the applications require in practice, and how much storage is allocated by the middleware.

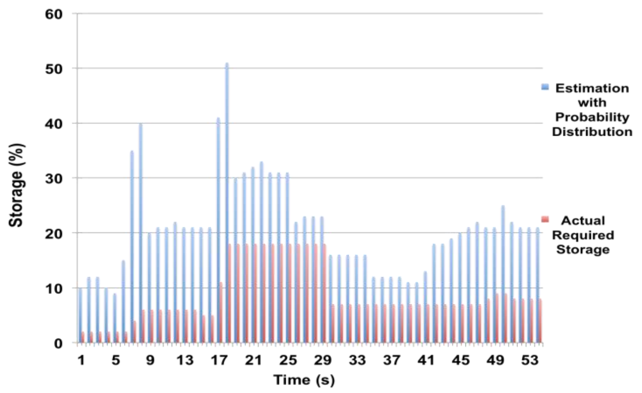

Fig. 3. Estimated vs. required storage at node level

Fig. 3 shows the average actual storage needed by the applications (red bars) vs. the estimated storage by middleware (blue bars). The vertical axis denotes the amount of node total memory in percentage.

Fig. 4 illustrates the same experiment on the cluster level, denoting how much collaborative storage inside the cluster-head is needed by applications running on cluster members (red bars) vs. how much storage is allocated to them (blue bars). A quick analysis of the figures shows the accuracy of estimation increases over time as more data become available.

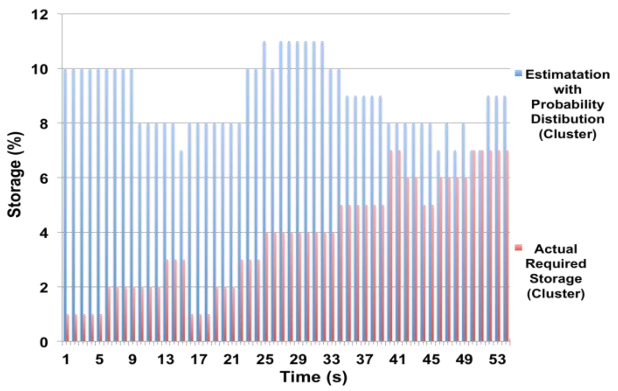

Fig. 4. Estimated vs. required storage in cluster-heads

Fig. 5 represents the overall error rate between the actual and estimated storage requirements in Fig. 3 and Fig. 4. As it shows, in total, the error rate is less than 15% in nodes and less than 10% in the cluster-heads. The lower error rate in the cluster-head is due to wider availability of data from multiple data sources (cluster members). There are sudden rises at instances 8 and 18 seconds on node level. These anomalies correspond to the first and second estimation periods when the required storage is being calculated. This abnormal behavior only happens at those periods.

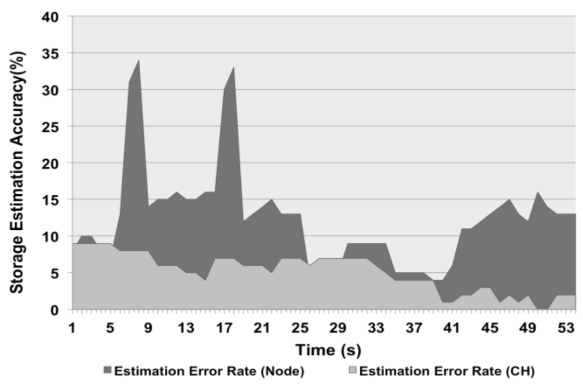

Fig. 5. Total error of node and cluster levels storage estimations

As we mentioned the overall error rate is much less in the cluster-heads due to availability of more data from multiple data sources. Therefore cluster density has an impact on the accuracy. Our next experiment shows how cluster-density affects accuracy of estimation function.

Fig. 6 shows how estimation accuracy varies based on the number of nodes in a cluster. The blue line shows the average estimation accuracy of a cluster-head with 1-38 members in simulation. The red line, on the other hand, shows the same experiment on a real WSN built from Sun Spot nodes. Based on our experiment 60% accuracy is the minimum level required for real-time applications as the error rates increases dramatically below this measure. Therefore we only carried on the experiment for another 20% extra below the minimum, which represents 32 nodes in simulation. In the case of our Sun Spot WSN, since we only had 14 nodes available we couldn't carry on the experiment for a higher number of nodes. However in this experiment, a cluster of 11 Sun Spots hit the minimum and therefore we achieved our objective.

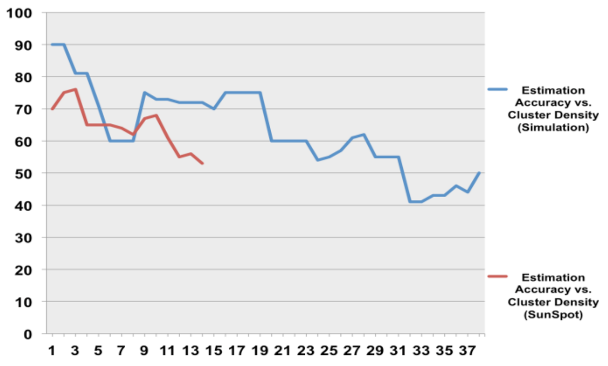

Fig. 6. Cluster density vs. estimation accuracy

As was explained in the previous section, allocations are only done when the probabilities are higher than the confidence threshold of 60%. As we explained, this threshold was selected based on a number of experiments. Fig. 6 shows the average accuracy of different probability results, both in simulations (red bars) and on the Sun Spots (blue bars). As this figure shows, probabilities less than 10% have negative accuracy, which in this case means allocated storage is way less than what is needed by the applications.

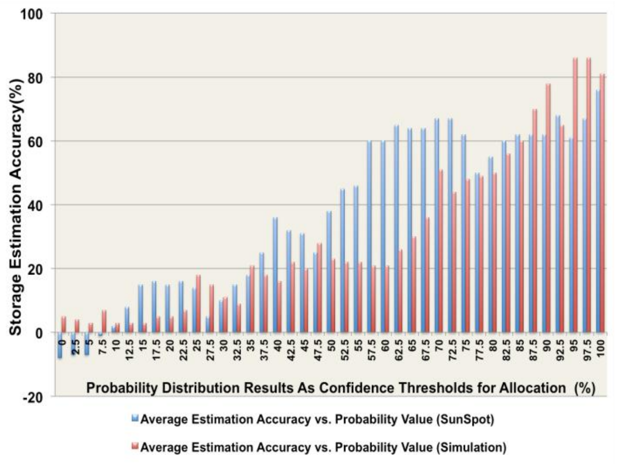

Fig. 7. Confidence thresholds vs. accuracy

Based on Fig. 7, we achieved our desired 60% accuracy with probability threshold values of 57.5% (0.575) and over, running on 14 Sun Spots. In simulation however, we achieved way less accuracy with 57.5% confidence threshold resulting in only 20% accuracy with 14 virtual nodes. This is due to wider distribution of events in the simulation environment and randomness of sensed variables. Therefore for the simulation, we based our confidence threshold on 85%. We repeated the first experiment, this time with 500 virtual nodes. (Fig. 3 and Fig. 4). Based on this figure, increasing the number of nodes improved our results by 50% on average, which is slightly on par with the Sun Spot confidence values in Fig. 6. In summary, such anomalies can be distributed to the way events are scattered in virtual environments in SXCS simulator.

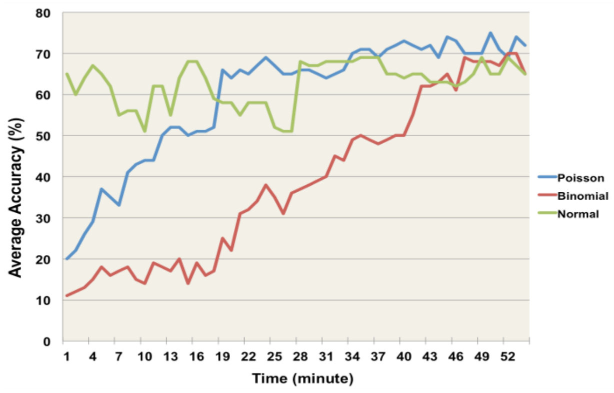

Fig. 8. Different distributions and their accuracies

Fig. 8 shows the comparison achieved from Poisson (blue line), Binomial (red line) and Gaussian distribution (green line). In general Poisson probability distribution proved to have higher accuracy over time compared to the binomial. This is worth noting that we only used Gaussian distribution to calculate the variance, in order to find the deviation of other two distributions for reserving extra storage.

Finally in our last experiment we repeated a style of experiment that we first reported in [2], to calculate how much longer processing an agent takes with the introduction of estimation functions in our architecture. Fig. 9 shows the processing time with (brown line) and without (orange line) estimation function with 30 concurrent applications running on the Sun Spots. Red and blue lines on the other hand represent the processing time with and without the estimation function respectively, while running 100 concurrent applications in the simulation. Estimation function proved to have improved Sensomax's response time to the agents and application dynamic demands. However, It is worth mentioning that this is also partially due to the reconfiguration of our architecture which reduces the number sequences an agents needs to go through before being fetched by the storage module. In order to implement the probability analysis, some of the extra processing for storing and indexing data was removed. As a result we achieved better performance using distribution techniques. It also depends on the nature of the application. For the purpose of this experiment we have used very lightweight event-driven and time-driven applications, which in overall has reduced the amount of processing by 30-45%.

IV. Conclusion

In this paper we have shown how WSNs can dynamically adapt to the different storage requirements of multiple concurrent applications using different probability distribution modeling. Binomial, Poisson and Gaussian probability distributions have been used in a uniform combination to present a mechanism for estimating the required storage per application, in both active and proactive manners. We constructed such a mechanism in Java, and utilized it as a computational module in Sensomax's component-based architecture. This led to lower agent processing time on the node level, as well as easing the collaborative storage on the cluster-level. Our future works will include game theory and market-based techniques for autonomous storage allocation.

References

[1] M. Haghighi and D. Cliff, "Sensomax: An Agent-Based Middleware For Decentralized Dynamic Data-Gathering in Wireless Sensor Networks," in *Proc. The 2013 International Conference on Collaboration Technologies and Systems*, CTS 2013, May 2013.

[2] M. Haghighi and D. Cliff, "Multi-Agent Support for Multiple Concurrent Applications and Dynamic Data-Gathering in Wireless Sensor Networks," in *Proc. Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, IMIS-2013, July 2013.

[3] *MICA2 Wireless Measurement System Product Information*. Crossbow Technology Inc., Crossbow. [Online]. Available: http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf

[4] P. Levis and D. Gay, *TinyOS Programming*, CUP, 2009.

[5] *Sun Spot Programmer's manual*, Oracle, Release v6.0, Sun Labs, Oracle, 2010.

[6] E. Upton and G. Halfacree, *Raspberry Pi. USER GUIDE IS*, John Wiley and Sons, 2012.

[7] S. Tilak, N. Abu-Ghazaleh, and W. B. Heinzelman, "Storage management in wireless sensor networks," *Mobile Wireless and Sensor Networks: Technology, Applications and Future Directions*, IEEE/Wiley, 2006, pp. 257–281.

[8] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-centric storage in sensornets with ght, a geographic hash table, Mobile Networks and Applications (MONET)," *Journal of Special Issues on Mobility of Systems, Users, Data, and Computing*, 2003.

[9] W. Liao and H. Yang, "An energy-efficient data storage scheme in wireless sensor networks," *Network Operations and Management Symposium (NOMS)*, 2012 IEEE.

[10] Z. Wang, K. Yang, and D. K. Hunter, "Modelling and analysis of multi-sink wireless sensor networks using queuing theory," in *Proc. 4th Conference on Computer Science and Electronic Engineering (CEEC)*, Sept. 2012, pp. 169-174.

[11] S. Aldaahmeh and M. Ghogho, "Traffic estimation for MAC protocols in distributed detection wireless sensor networks," in *Proc. the 20th European Signal Processing Conference (EUSIPCO)*, August, 2012, pp. 719-723.

[12] R. Ranjan and S. Kar, "A novel approach for finding optimal number of cluster head in wireless sensor network," in *Proc. National Conference on Communications (NCC)*, January 2011, pp. 1-5.

[13] R. Niu and P. K Varshney, "Decision fusion in a wireless sensor network with a random number of sensors," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, March 2005, pp. 861-864.

Mo Haghighi is a doctoral researcher at the University of Bristol, UK. He is currently pursuing his research in the area of "Decentralized Agent-based Adaptive Dynamic Data Gathering in Large-scale Wireless Sensor Networks". He has obtained a BEng in Electronic and Telecommunications engineering followed by an MSc in Wireless Sensor Networks. He began his research in a joint collaboration between the University of Bristol, the BAE Systems and Large-Scale Complex IT Systems (LSCITS). Prior to his PhD, he had worked for Sun Microsystems/Oracle for over two years, primarily involved in academic projects. As a member of LSCITS, his research has broadened to include complexity science, Cloud computing, multi-agent systems and quantitative data analysis for large-scale complex systems. Mo has extensive programming experience in Java, C++ and Assembly. He also specializes in designing embedded systems (ARM, Freescale, Microchip and Intel), large-scale distributed systems, network security, machine learning, LoWPANs and Microwave communications.