Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria di Modena

Corso di Laurea Specialistica in Ingegneria Informatica

OPTIMIZATION OF CONTINUOUS-MONITORING QUERIES IN SENSOR NETWORKS

Relatore:
[Chiar.ma] Prof.ssa Sonia Bergamaschi

Candidato:
Alessio Cavallini

Anno Accademico 2005/2006

Key words:

Sensor Networks
Continuous-monitoring
Quantile
Time-series
Holt-Winter
Forecast
Threshold

Acknowledgments

Professor Sonia Bergamaschi for her constant support during this research, but most of all for having given and backed the great opportunity to spend three months doing research at the AT&T Shannon Laboratories.
Divesh Srivastava and Graham Cormode for having advised my activity on sensor networks at the AT&T.
My parents, for having taught me that whenever you learn it's you to grow wiser, nobody else.
Giada and all my friends for the great support they give me during the months spent away from home.

Alessio

The Internet is an elite organization,
most of the population of the world
has never even made a phone call.

Noam Chomsky

SINTESI

La ricerca descritta in questa tesi è stata svolta presso gli AT&T Shannon Laboratories, Florahm Park, NJ (USA), divisione Database, sotto la supervisione di Graham Cormode, Sr Inventive Research Specialist, e Divesh Srivastava, responsabile della divisione Database. Lo scopo di questa tesi è quello di ottimizzare le query di monitoraggio continuo nelle reti di sensori (Sensor Network, SN). Il concetto di rete di sensori si è ammodernato nel tempo, non si deve più pensare alle reti di sensori solo come quelle reti formate da sensori di monitoraggio di condizioni ambientali, ma il significato di SN si è ampliato fino a comprendere i programmi che effettuano monitoraggio all'interno di router e server, in questo caso avremmo una rete di sensori formata da sensori software invece che hardware.. Le reti di sensori di solito sono usate per effettuare un monitoring continuo dell'ambiente in cui sono installate, che può andare dalla temperatura di un edificio al numero di connessioni dei server della rete. Una specializzazione delle reti di sensori sono le reti di sensori wireless, Wireless Sensor Network (WSN), in questo tipo di reti i dispositivi sono tipicamente alimentati tramite una batteria, e comunicano tramite una rete senza fili, non necessariamente secondo standard 802.11, ma possono usare anche infrarossi, Bluetooth o GSM. Questo particolare tipo di rete è posizionato tipicamente in ambienti complessi dove il cablaggio non è possibile e risulta difficile anche la normale manutenzione. I dispositivi che compongono le reti di sensori si dividono in nodi e coordinatore, i primi sono i dispositivi che effettuano la rilevazione, il coordinatore è quel dispositivo che tiene traccia dei valori rilevati dai nodi che coordina e risponde alle query effettuate che riguardano i nodi sotto il suo controllo. La struttura della rete tipicamente è fissata a priori e gerarchica, e ogni dispositivo ha un ruolo prefissato, chi nodo e chi coordinatore e non possono cambiare,ma sono possibili strutture molto più complesse che comprendono anche algoritmi di routing dinamici e nodi che posso assumere lo status di coordinatore al cambiare delle condizioni interne della rete.

L'ottimizzazione delle query si rende necessaria nelle reti di sensori cablate perché il monitoring continuo, per mantenere i dati del coordinatore aggiornati, genera un overhead elevato, soprattutto se le reti non fanno un osservazione periodica, ma rilevano

eventi, quindi all'aumentare del numero di eventi aumentano anche il numero di comunicazioni che i nodi che i nodi devono effettuare al coordinatore per mantenere i dati aggiornati in modo che si possa rispondere alle query effettuate. Nelle WSN il problema è ancora più complesso, poiché i dispositivi sono alimentati tramite batteria, la comunicazione wireless richiede molta energia, infatti l'energia necessaria per mandare un bit di dati usando tecnologia wireless è la stessa usata per compiere 800 operazioni. In questo caso diminuire le comunicazioni effettuate dai nodi al coordinatore è una condizione necessaria per permettere un'autonomia sufficiente ai sensori.

Di seguito alcuni esempi di tipiche query effettaute su reti di sensori:

- Monitorare la temperatura media, utilizzando una media mobile di 1 ora, dell'ambiente monitorato
- Monitorare la temperatura di ogni sensore se e solo se la temperatura media supera un valore prefissato.
- Monitorare il numero di connessioni che arrivano al nodo A.
- Monitorare le dimensioni dei file richiesti nelle connessioni http.

Se il numero di connessione effettuate alla rete supera un valore prefissato eseguire un monitoring della rete rilevando se il numero di connessioni ricevute da un singolo nodo supera un altro valore prefissato.

Si consideri un ambiente distribuito formato da un certo numero k di dispositivi remoti, i flussi di dati arrivano continuamente ai nodi, mentre il coordinatore è responsabile di generare le risposte approssimate alle query poste. Il modello di stream processing è lo stesso utilizzato nei modelli presentati da Cormode et al. [**7**][**10**], dove nessuna comunicazione diretta fra i nodi è permessa. Ad ogni sito remoto $j \in \{1,\ldots,k\}$, viene rilevato un flusso di dati S_j , i cui dati appartengono ad un dominio $[U] = \{0,\ldots, U\text{-}1\}$, per esempio nel caso di IP monitoring all'interno dei router $[U]$ è un dominio di 64-bit, formato dalle coppie sorgente destinazione, e S_j cattura la frequenza di una specifica coppia osservata dal router j. Il principale problema è rispondere alle query sulla distribuzione dell'insieme dei flussi $S = U_j\ S_j$ generata nel dispositivo coordinatore. La dimensione di ogni distribuzione di frequenza che i nodi rilevano è dipendente dal numero di aggiornamenti rilevati nel lasso di tempo considerato, se si considera di fare monitoring all’interno di un NOC (Network Operation Center) il numero di aggiornamenti, anche in un lasso di tempo molto piccolo è molto alto, e la dimensione della distribuzione è molto grande; questo alza notevolmente il costo della comunicazione, quindi la distribuzione deve essere scambiata tra nodo e coordinatore usando una forma complessa utilizzando i quantili. I quantili delle rilevazioni prese ad intervalli regolari lungo la funzione di distribuzione cumulativa di una variabile casuale, permette facilmente di ottenere valore minimo, massimo e mediana della distribuzione. Caratteristica fondamentale dei quantili è il rank, che viene trasmesso insieme ai valori dei quantili per formare la distribuzione, definito come $r(v) = |\{u \in S : u \leq v\}|$. Per ottimizzare l'esecuzione delle query di monitoraggio continuo all'interno di reti di sensori mi sono focalizzato sul tracciamento dei quantili del flusso globale *S*. Questo tracciamento è intrinsecamente approssimato, poiché i quantili comprimono la distribuzione prendendo solo un certo numero di valori con il loro rank, quindi si perdono tutti i valori che stanno tra due valori selezionati, l'approssimazione minima è data dal "passo" con cui si prelevano i quantili, per esempio se si sceglie di avere 50 quantili che rappresentano la distribuzione, l'errore minimo ammesso è del 2%. Questa approssimazione è la minima possibile e non permette di risparmiare notevolmente sul numero di comunicazioni, ma solo sulla dimensione. Si può aumentare l'approssimazione consentita, avendo comunque la garanzia di avere risposte alle query poste entro i margini richiesti. Cormode et al. [**7**][**10**] hanno presentato una soluzione per il tracciamento dei quantili, basata sul monitoring di flussi di dati, e quindi di distribuzioni, stabili nel lungo periodo e che non comprendevano sottrazioni. Queste condizioni si verificano per esempio se mi monitorizza una rete senza considerare dei periodi, che siano minuti giorni o ore, ma solo il numero delle connessioni e le dimensioni dei dati richiesti a partire da un punto iniziale in poi. La sola addizione di valori rende molto stabile nel tempo la distribuzione dei valori e quindi la rende predicibile. Ho deciso invece di ottimizzare l'esecuzione di query di monitoraggio continuo in ambienti in cui le distribuzioni sono relative al tempo e quindi cambiano notevolmente durante l'osservazione. Essenzialmente è possibile eseguire due tipi di monitoring dei dati, uno fatto a periodi prefissati di tempo, per esempio orario o giornaliero, e l'altro effettuato con l'utilizzo di una finestra scorrevole, per esempio della lunghezza di un ora con sessanta intervalli lunghi un minuto al suo interno. Queste distribuzioni di valori non sono stabili, poiché sono affette da trend e stagionalità in maniera molto più evidente che in quelle puramente additive, per questo sono necessari nuovi metodi per predire le distribuzioni, più complessi di quelli utilizzati da Cormode et al.

Stabilendo un margine di errore tollerabile è a questo punto possibile riuscire a ridurre il numero di comunicazioni cercando di far predire al coordinatore i valori misurati dai nodi. Se la predizione effettuata dal coordinatore è accurata entro i margini di errore imposti nella query, nessuna comunicazione è effettuata dai nodi al coordinatore, in caso contrario i dispositivi inviano un aggiornamento.

Per rispondere alle query di monitoraggio continuo sono necessarie due informazioni, che devono poi essere predette da parte del coordinatore per riuscire ad ottimizzare l'esecuzione delle query, che sono:

- Numero di aggiornamenti ricevuti nel periodo di tempo considerato
- Quantili selezionati e relativi rank

La prima informazione racchiude anche le informazioni sulla soglia che viene computata al lato coordinatore, poiché normalmente le imposizioni di soglia sono fatte sulla distribuzione globale e non su quelle locali dei siti remoti.

Si può considerare il numero di aggiornamenti ricevuti da ogni sito remoto come una time-series. Una serie temporale è una sequenza di dati presi ad intervalli regolari, questa definizione rispecchia esattamente quello che è il numero di aggiornamenti, cioè un valore di un osservazione preso ad intervalli regolari, sia che parliamo di periodi di tempo che di finestre scorrevoli. Le serie temporali mettono a disposizioni strumenti anche complessi per la predizione dei valori della serie temporali nel futuro. Dopo aver considerato alcuni modelli di predizioni ho deciso di concentrarmi su il modello di predizione di Holt-Winter che ha la caratteristica di considerare oltre al valore precedente anche il trend e la stagionalità. Questo modello si avvale di tre parametri per far pesare in maniera diversa i tre componenti: valore precedente, trend e stagionalità, nella composizione del valore predetto; per la selezione di questi parametri ci si avvale del metodo del "Minimal Mean Square Value" che è su dati campione e cerca di selezionare i parametri che minimizzano la distanza fra il valore predetto e il valore reale nella serie presa in esame che serve per fare training. Questo metodo si è rivelato inadatto all'utilizzo che se ne deve fare, poiché le query impongono uno stringente margine di errore, e se la previsione supera il margine di errore imposto, il nodo deve comunicare al coordinatore il valore rilevato dato che il valore predetto dal coordinatore è errato, e quindi non ha importanza di quanto sia vicino il valore predetto a quello reale se il predetto supera il margine d'errore prefissato. Ho quindi elaborato un algoritmo che sceglie la terna di parametri migliori in base alla minimizzazione del numero di errore commesso dall'algoritmo di Holt-Winter usando i parametri selezionati. Per migliorare ancora l'algoritmo ho introdotto un fattore di correzione d'errore che si basa sulla considerazione che due valori lontani una stagione l'uno dall'altro abbiano la stessa distanza, in percentuale, dal valore che li precede; anche questo termine è preceduto da un parametro che ne decide il peso all'interno del valore predetto. Con l'utilizzo di questi due accorgimenti ho ottenuto il 25% in meno di errori rilevando il numero di aggiornamenti a periodi prefissati, del 45% usando la finestra scorrevole.

Per monitorare l'andamento della distribuzione di frequenza, il coordinatore deve tenere traccia dei valori dei quantili e del loro rank. In particolare il rank dell'ultimo valore è il numero totale di valori della distribuzione, quindi il numero di aggiornamenti, e il rank del primo valore non è necessariamente zero poiché per la definizione di rank vengono conteggiati anche il numero di valori uguali a quello di cui si cerca il rank.

La predizione corretta del rank è fondamentale per ottimizzare le query, infatti l'errore viene misurato sul rank predetto, il rank del valore reale nella distribuzione di frequenza predetta dal coordinatore deve essere contenuto nel margine di errore prefissato dalla query. Se ciò non dovesse succedere per prima cosa il nodo prova a ricalcolare il rank partendo dal numero di aggiornamenti reali e non predetti ricevuti, se in questo caso il margine di errore è rispettato, il nodo invia al coordinatore solo il numero di aggiornamenti reale; se non è soddisfatto il vincolo d'errore, il nodo invia una distribuzione di frequenza minimizzata usando quantili e rank. Per calcolare il rank predetto ho sviluppato un algoritmo più sofisticato di quello usato da Cormode et al. che ha permesso di ridurre il numero di comunicazioni che comprendevano anche i sommari di quantili e rank in media del 70% e quelle che coinvolgono solo gli aggiornamenti ricevuti dal nodo del 10%.

La gestione di soglie globali in un ambiente distribuito non è banale, infatti il coordinatore deve sempre sapere se il valore totale rilevato dalla somma dei nodi si avvicina alla soglia, e se viene superata si devono intraprendere certe azioni. La soglia è stata introdotta poiché non vi era alcun interesse a monitorare certi processi se rimanevano stabili sotto un determinato valore globale e si preferiva tentare di tenere il numero di comunicazioni basso effettuando monitoraggio solo nel caso servisse realmente. Per esempio, non vi è alcun interessa a tenere traccia del numero di auto che percorrono l'autostrada a meno che il numero totale di auto non superi un valore critico. Per effettuare questo all'interno di un sistema distribuito come le reti di sensori mi sono avvalso di un algoritmo per il monitoraggio del "total count" sviluppato da Kevin Yi [34] presso gli AT&T Shannon Labs, e con la sua collaborazione ho esteso il suo algoritmo iniziale, pensato solo per processi additivi, per supportare le sottrazioni e quindi anche le finestre scorrevoli, integrandolo poi con l'algoritmo di Holt-Winter, modificato con il correttore di errore, per fornire risposte alle query di monitoring continuo con all'interno delle soglie. L'algoritmo si basa su delle soglie temporanee che sono inviate dal coordinatore ai nodi, ogni volta che un nodo supera la soglia temporanea lo riporta al coordinatore, quando sono state supertate un numero di soglie temporanee equivalenti al numero di nodi, una nuova soglia temporanea, la metà della precedente, viene inviata ai nodi, facendo proseguire l'algoritmo. Se il numero di aggiornamenti rilevato dai nodi è inferiore alla soglia temporanea precedente, il nodo manda al coordinatore un aggiornamento negativo che serve per gestire le finestre scorrevoli e far arretrare la soglia temporanea: se il numero di aggiornamenti arrivati al coordinatore è negativo ma in valore assoluto uguale al numero dei nodi. Il problema delle soglie è ancora aperto, in quanto sono possibili miglioramenti all'algoritmo che tengano in considerazione la possibile oscillazione costante di un valore intorno la soglia temporanea che causa un numero di comunicazioni notevoli fra nodo e coordinatore.

Contents

List of figures

List of tables

1. INTRODUCTION

I write this thesis during my internship at the AT&T Shannon Laboratories, Florahm Park, NJ (USA), my advisors during that period were Graham Cormode and Divesh Srivastava, two high level researchers in databases.

1.1 Sensor Networks and Wireless Sensor Networks

1.1.1 Definition

A sensor network is a set of small autonomous systems, called sensor nodes which cooperate to solve at least one common application; their task include some kind of perception of physical parameters[2].
A wireless sensor network (WSN) is a wireless network consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at different locations[1][2].
First research on wireless sensor networks was mainly motivated by military applications, with DARPA continuing to found a number of prominent research projects that are commonly regarded as the cradle of sensor network research. The type of applications considered by these projects led to a de facto definition of wireless sensor

network as a large-scale (thousand of nodes, covering large geographical areas), wireless, ad hoc, multi-hop, unpartitioned network of homogeneous, tiny, mostly immobile (after deployment) sensor nodes that would be randomly deployed in the area of interest. More recently, other civilian application domains of wireless sensor networks have been considered, such as environmental and species monitoring, agriculture, production and delivery, healthcare, etc… Concrete projects targeting these applications areas indicate that the above definition of a wireless sensor networks does not necessary apply for these applications; networks may consist of heterogeneous and mobile sensor nodes, the network topology may be as simple as a star topology and networks may make use of existing communication infrastructures[1][3].
Unique characteristics of a WSN are:

- Small-scale sensor nodes
- Limited power they can harvest or store
- Harsh environmental conditions
- Node failures
- Mobility of nodes
- Dynamic network topology
- Communication failures
- Heterogeneity of nodes
- Large scale of deployment
- Unattended operation

1.1.2 The node

In addition to one or more sensors, each node in a sensor network is typically equipped with a radio transceiver or other wireless communications device, a small microcontroller, and an energy source, usually a battery. Depending on the actual needs of the application, the form factor of a single sensor node may vary form the size of a shoe box (e.g., a weather station) to a microscopically small particle (e.g., for military applications where sensor nodes should be almost invisible). Similarly the cost of a single device may vary from hundreds of Euros , for networks of very few, but powerful nodes, to few cents, for large-scale networks made up of very simple nodes.

Since sensor nodes are untethered autonomous devices, their energy and other resources are limited by size and cost constrains. Varying size and cost constrains directly result in corresponding varying limits on the energy available, as well as on computing, storage, and communication resources. Hence, the energy ad other resources available on a sensor node may also vary greatly from system to system. Power may be either stored (e.g., batteries) or scavenged form environment (e.g., by solar cells).

1.1.3 Communication Modality

For wireless communications among sensor nodes, a number of communication modalities can be used such as radio, diffuse light, laser, inductive and capacitive coupling, or even sound, Perhaps the most common modality is the radio waves, since these do not require a free line of sight, and communication over medium ranges can be implemented with relatively low power consumption and relatively small antennas. Using light beams for communication requires a free line of sight and may interfere with ambient light and daylight, but allows for much smaller and more energy-efficient transceivers compared to radio communication. Smart Dust [4], for example, uses laser beams for communication. Inductive and capacitive coupling only works over small distances, but may be used to power a sensor node. Most passive Radio Frequency Identification (RFID) systems use inductive coupling, for example, sound and ultrasound is typically used for communication under water or to measure distances based on time-of-flight measurements. Sometimes, multiple modalities are used by a single sensor network system. The communication modality used obviously influences the design of medium access protocols and communication protocols, but also affects other proprieties that are relevant to the application[1]. The various communication modalities can be used in different ways to construct an actual communication network. Two common forms are so-called infrastructure-based networks on the one hand and ad hoc networks on the other hand. In infrastructure-base networks, sensor nodes can only directly communicate with so-called vase station devices. Communication between

sensor nodes is relayed via the base station. If there are multiple vase stations, these have to be able to communicate with each other. The number of the base stations depends on the communicate range and the area covered by the sensor nodes. Mobile phone networks, and Smart Dust [4] are examples of this type of network.
In ad hoc networks, nodes can directly communicate with each others without an infrastructure, Node may act as routers forwarding messages over multiple hops on behalf of other nodes.
Since the deployment of an infrastructure is a costly process, and the installation of an infrastructure may often not be feasible, ad hoc networks are preferred of many applications. However an infrastructure is already available anyway (such as the GSM network), it might be used for certain sensor network applications. Combination of ad hoc networks and infrastructure-base networks are sometime used, where cluster of sensor nodes are interconnected by a wide area infrastructure-based network.

1.1.4 Energy Problem

The aspect of energy supply was not contained in the initial definition. Some nodes are powered with solar cells rather than with batteries. However, at the moment the major effort of sensor network research goes into the field of energy and communication efficiency which is also the focus of this thesis. Features like computational power and memory per unit square have been developing much faster than advances in battery technology.

Battery technology	Watt hours/gram
Lithium-Ions in chemical batteries	0,3
Methanol in fuel cells	3
Tritium in nuclear batteries	850
Polonium-210 in nuclear batteries	57000

Table 1.1: Future energy potential for different battery technologies.

Table 1.1 shows an overview of the fitness of different matters as energy source. Lithium-ion batteries among the best options today. It can be expected that a battery of about 1 gram deliver 1 watt of energy for 0.3 hours (18 minutes). Fuel cells have potential that is about 10 times higher but some problem to be solved, especially when it comes to miniaturization. Nuclear batteries are among potential technologies that carry the greatest potential with about 57,000 watt hours/gram [2].

1.2 Database Technologies

Database technologies are beginning to have a significant impact in the emerging area of wireless sensor networks. The WSN community has embraced declarative queries as a key programming paradigm for large set of sensors [6]. In the emerging industrial arena, one of the leading vendor (Crossbow) is bundling a query processor with their devices. Declarative querying has proved powerful in allowing programmers to task an entire network of sensor nodes, rather than requiring them to worry about programming individual nodes. However the metaphor that "the SN is a database" has proved misleading [6]. Databases are typically treated as complete, authoritative sources of information; the job of a database query engine has traditionally been to answer a query correctly based upon all the available data. Applying this mindset to SNs results in two problems:

- Misrepresentation of data: In the SN environment, it is impossible to gather all the relevant data. The physically observable world consist f a set of continuous phenomena in both time and space, so the set of relevant data is in principle infinite. Sensing technologies acquire sample of physical phenomena at discrete points in time and space but the data acquired by SN is unlikely to be random sample of physical process, for a number of reasons: non uniform placement of sensors in space, faulty sensors, high packet loss rate, etc… so a straightforward interpretation of SN readings as a database may not be a reliable representation of the real word.

- Inefficient approximate queries: Since a SN cannot acquire all possible data, any reading from SNs is approximate, in the sense that it only represents the true state of the world at the discrete instants and locations where the sample were acquired.

1.3 Generalization of Sensor Networks

Usually we consider as sensor nodes only small, more or less, electronic devices that have the capability to measure some physical phenomena, but if we think about the ITC world and expand the meaning of measuring a physical phenomena to measure something that could happen, we find a complete new family of sensor networks. For example the routers: a router is connected with almost another router, every router have common duty, the packet routing, and every router can measure something, for example, the number of packet directed to an host, or the size of every packet; so a router network is a sensor network.
These new kinds of sensor networks have one big problem in common with the traditional sensor networks: the cost of the communications. In WSN to communicate with the coordinator of the networks or with another node is a problem for the energy cost of the communication, because “transmitting a single bit of data is equivalent to 800 instructions” [5], in these new kinds of sensor networks the communication is a big problem for the overhead of bandwidth. If we measure network related events, growth in the number of events cause more communications between the nodes or between node and coordinator and so the risks of a collapse of the network increases.

1.4 Distributed Tracking

With traditional database systems to optimize for performance on one-shot queries, emerging large-scale monitoring applications require continuous tracking of complex

aggregates and data distributed summaries over collections of physically distributed streams. Thus, effective solutions have to be simultaneously space efficient (at each remote site), communication efficient (across the underlying communication network) and provide continuous guaranteed-quality estimates. Traditional data-management applications such as managing sales records, transactions, inventory, or facilities typically require database support for a variety of one-shot queries, including lookups, sophisticated slice and dice operations, data minig tasks, and so on. One-shot means the data processing is essentially done once, in response to the posed query. This has led to an enormously successful industry of database optimized for supporting complex, one-shot SQL queries over large amounts of data [10].
Recent years, however, have witnessed the emergence of a new class of large-scale event monitoring applications that pose novel data-management challenges. In one class of applications, monitoring a large-scale system is an operational aspect of maintaining and running the system. As an example, consider the Network Operations Center (NOC) for the IP-backbone network of a large ISP. Such NOCs are typically impressive computer facilities, monitoring hundreds of routers, thousands of links and interfaces, and blisteringly fast set of events at different layers of the network infrastructure, ranging from fiver-cable utilizations to packet forwarding at routers, to VPNs and higher –level transport constructs. The NOC has to continuously track patterns of usage levels in order to detect and react to hot spots and floods, failures of links or protocols, intrusions, and attacks. A similar example is that of data centers and web-content companies, such as Akamai, that have to monitor access to the thousand of web-caching nodes and do sophisticated load balancing, not only for better performance but also to protect against failures. Similar issues arise for utility companies such as electricity suppliers that need to monitor he power grid and customer usage. A different class of applications is one in which monitoring is the goal in itself. For instance, consider a wireless network of seismic, acoustic, and physiological sensors that are deployed for habitat, environmental, and health monitoring. Here, the sensor systems monitor the distribution of measurements or trend analysis, detecting moving objects, intrusions or other adverse event. Similar issues arise in sophisticated satellite-base systems that do atmospheric monitoring for weather patterns [7].

Examining these monitoring applications in detail allows to abstract a number of common elements:

- Monitoring is continuous; we need real-time tracking of measurements or events, not merely one-shot responses to sporadically posed queries;
- Monitoring is inherently distributed; the underlying infrastructure comprises several remote sites, each with its own local data source that exchanges information through a communication network;
- There typically are important communication constraints owing to network-capacity restrictions (e.g. in IP-network monitoring, where collected utilization and traffic is voluminous [8]) or power and bandwidth restrictions (e.g. in WSN, where the communications overhead is the key factor in determining sensor battery life [9]).

Furthermore, each remote site may see a high-speed stream of data and has its own local resource constraints, such as storage-space or CPU-time constraints. This is true for IP routers that cannot possibly store the log of all observed traffic due to the ultra-fast rates at which packets are forwarded. This is also true for wireless sensor nodes, even though they may not observe large data volumes, since they typically have very small memory onboard [10].

1.5 Approximation

There are two key aspects of large-scale monitoring problems:

- One needs a way to effectively monitor the complete distribution of data (e.g., IP traffic or sensor measurements) observed over the collection of remote sites. Having an accurate picture of the overall data distribution is crucial in understanding system behavior and characteristics, tracking important trends, ad making informed judgments about measurements or utilization patterns. In other word, while hardwired outlier detection methods can be of use for certain applications (e.g., network anomaly detection), monitoring the entire data distribution give as much broader and more robust indicator of overall system behavior; such indicator are critical, for instance, in network-provisioning systems that try to provision routing paths with guaranteed Quality-of-Service parameters over an IP network.

- Answers that are precise to the last decimal are typically not needed when tracking statistical properties of large-scale systems; instead approximate estimates, with reasonable guarantees on the approximation error, are often sufficient, since we are typically looking for indicators or patterns, rather than precisely-defined events [7]. Obviously, this can work in our favor, allowing us to effectively tradeoff efficiency with approximation quality.

The focus is on large-scale monitoring problems that aim to continuously provide accurate summaries of the complete data distribution over a collection of remote data streams. Solution for such monitoring problems have to work in a distributed setting (i.e., over a communication network), be real-time or continuous, and be space and communication efficient; furthermore, approximate, yet accurate, answer suffice.

1.6 Continuous-monitoring Queries

The typical queries inside a distributed environment such as the Sensor Networks or the Wireless Sensor Networks are:

- Monitor the total number of connection at router A.
- Monitor the total number of connection inside the network.
- The average size of the file requested at server A.
- The average temperature in the last 24 hours in the monitored environment.
- The size of the larger file downloaded in the network in the last 6 hours
- Monitor the number of connection in the network, and the size of those only if the number excides 40000

All those queries are continuous-monitoring queries, and require a lot from all the nodes for the answer in particular I the data have to be updated periodically , that is very typical in monitoring queries, because how send the queries usually wants close real time data. Optimizing the queries means make these less expensive, in particular in terms of communications cost, Because are very simple queries in terms of execution inside every node, but in most of case is very expensive communicate the read values from the observers to how needs the values observed. As I observed in the precedent paragraph, the real goal is the efficient monitoring of the distribution of data and not the precise decimal value so a good way for optimizing this kind of queries is to introduce error bounds in the queries such as:

- Monitor the total number of connection at router A with an approximation of 5%
- The average temperature ±0.2°C in the last 24 hours in the monitored environment.
- Monitor the number of connection in the network (5% error), and the size of those only if the number excides 40000 ± 200

Introducing an error there is a possibility to predict the observed values instead communicating the value to how made the queries.

The main objective is to construct mathematical algorithms to predict accurately the observed values, for reduce the communications cost inside the distributed environment, optimizing in this manner the query execution.

2. APPROXIMATE QUANTILES TRACKING

2.1 System Architecture

We consider a distributed-computing environment, comprising a collection of *k* remote sites and a designed coordinator site. Streams of data update arrive continuously at remote sites, while the coordinator site is responsible for generating the approximate answers to (possibly, continuous) user queries over the union of all remotely observed streams.

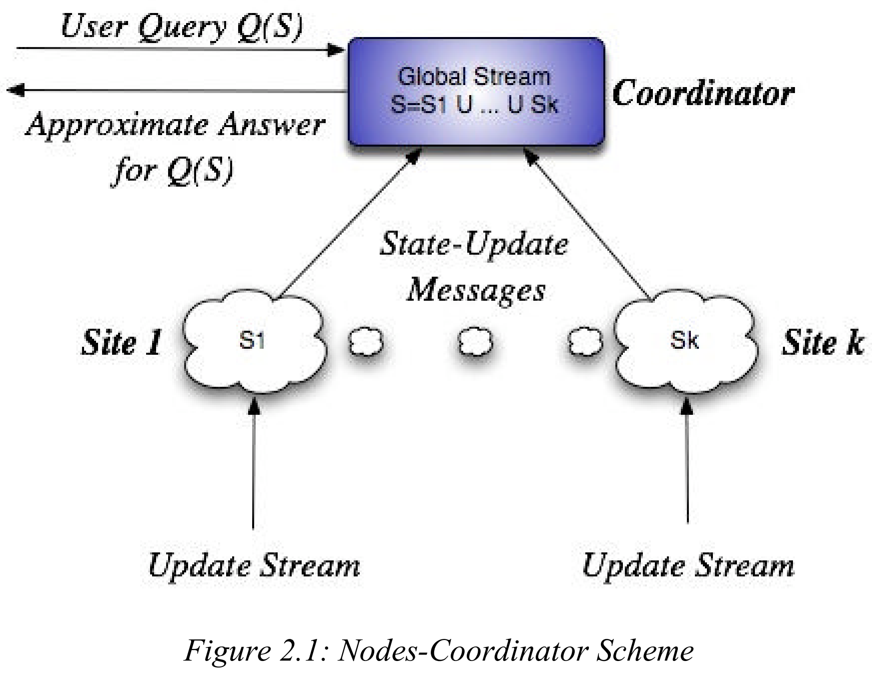

Figure 2.1: Nodes-Coordinator Scheme

The distributed stream-processing model used is the same of the model presented by Cormode et al [**7**], similar to that of Olston *et al*. [11][12] and Das et al. [13] where no direct communication between remote node is allowed; instead, as illustrated in figure *2.1*, a remote sire exchanged messages only with the coordinator, providing it with state information of its (locally-observed) streams.

That hierarchical processing model is representative of a large class of applications, including network monitoring where central Network Operation Center (NOC) is responsible for processing network traffic statistics collected from switches, router and/or Element Management Systems (EMSs) distributed across the network.

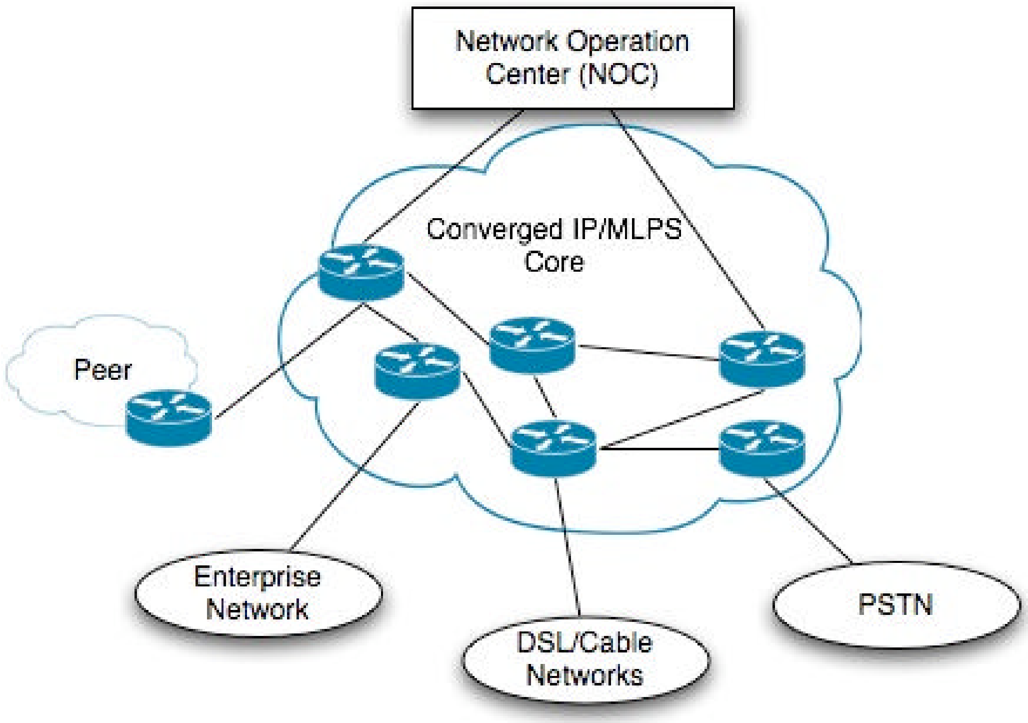

Figure 2.2: NOC and the monitored network

2.2 Streams

At each remote site $j \in \{1,...,k\}$, the local update stream renders a multi-set S_j (or, in other words, a frequency distribution) over data elements form integer domain $[U] = \{0,..., U-1\}$. As an example, in the case of IP routers monitoring the number of connections between source and destination IP address, $[U]$ is the domain of 64-bit (source, destination) IP-address pairs, and S_j capture the frequency of specific (source, destination) pairs observed at router j. [7] I use S_j to denote both the update stream at site j as well as the underlying multi-set/frequency distribution in what follows. Assume that each stream update at remote site j has the form $<+1,v>$, denoting the insertion of element $v \in [U]$ in the S_j multi-set (i.e., an increase of +1 in v's net frequency in S_j).

2.3 Continuous-monitoring queries

2.3.1 Quantiles

Quantiles are essentially points taken at regular intervals from the cumulative distribution function of a random variable. Dividing ordered data into q essentially equal-sized data subsets is the motivation for q-quantiles; the quantiles are the data values marking the boundaries between consecutive subsets. Put another way, the k^{th} q-quantile is the value x such that the probability that a random variables will be less than x is at most $\frac{k}{q}$ and the probability that a random variable will be less than or equal to x is at least $\frac{k}{q}$. There are $q - 1$ quantiles, with k an integer satisfying $0 < k < q$.

Some quantiles have special names:

- The 100-quantiles are called percentiles.
- The 10-quantiles are called deciles.
- The 5-quantiles are called quintiles.
- The 4-quantiles are called quartiles.

For an infinite population, the k^{th} quantile is the data value where the cumulative distribution function is equal to $\frac{k}{q}$. For a finite N sample size, calculate $N.\frac{k}{q}$, if this is not an integer, then round up to the next integer to get the appropriate sample number (assuming samples ordered by increasing value); if it is an integer then any value from the value of that sample number to the value of the next can be taken as the quantile, and it is conventional (though arbitrary) to take the average of those two values.

More formally: the *k*-th "*q*"-quantile of the population parameter *X* can be defined as the value "*x*" such that:

$P(X \le x) \ge p$ and $P(X \ge x) \ge 1 - p$ where $p = \frac{k}{q}$

If instead of using integers *k* and *q*, the *p*-quantile is based on a real number *p* with $0<p<1$ then this becomes: The *p*-quantile of the distribution of a random variable *X* can be defined as the value(s) *x* such that:

$P(X \le x) \ge p$ and $P(X \ge x) \ge 1 - p$

If a distribution is symmetrical, then the median is the mean (so long as the latter exists). But in general, the median and the mean differ; for instance, with a random variable that has an exponential distribution, any particular sample of this random variable will have roughly a 63% chance of being less than the mean. This is because the exponential distribution has a long tail for positive values, but is zero for negative numbers.

Quantiles are useful measures because they are less susceptible to long tailed distributions and outliers.

Empirically, if the data you are analyzing are not actually distributed according to your assumed distribution, or if you have other potential sources for outliers that are far removed from the mean, then quantiles may be more useful descriptive statistics than means and other moment related statistics [14].

2.3.2 Tracking Problem

The main problem is to effectively answer user query on the frequency distribution of the global collection of streams $S = U_j\ S_j$ at the coordinator site. Rather than one-time query evaluation, we assume a continuous–querying environment, which implies that the coordinator needs to continuously maintain (or track) a picture of the global frequency distribution S as the local update stream S_j evolve at individual remote sites. More specifically, the primary focus is on continuously tacking the quantiles of global frequency distribution S at the coordinator [10][7]. The distributed nature of the local stream S_j comprising the global frequency distribution S makes a very challenging problem. A naïve scheme that accurately tracks the quantiles of S by forcing remote sites to ship every remote stream update to the coordinator is clearly impractical, since it not only imposes an inordinate burden on the underlying communication infrastructure, especially for high-rate data streams and large numbers of remote sites, but also drastically limits the battery life of power constrained remote devices, such as wireless sensor nodes [6][9]. Instead, to reduce communication overhead, we focus on the continuous tracking of S's approximate quantiles at the coordinator site with strong guarantees on quality of approximation. This allows the schemes to effectively trade-off communication efficiency and quantile approximation accuracy in a precise quantitative manner; in other words, larger error tolerances for the approximate quantiles at the coordinator imply smaller communication overheads to ensure continuous approximate tracking.

More formally, let $N = |S|$ denote the total size of the global data stream S. For a domain value v $[U]$, we use $r(v)$ and $q(v)$ to denote the absolute rank and quantile (i.e., relative) rank of v in S, respectively; in other words, $r(v) = |\{u \in S : u \leq v\}|$ and $q(v) = \frac{r(v)}{N}$. As shown in figure 2.3 the rank of a value considers also the values equal to the given one. Given a prespecified error tolerance ε, the goal is to continue to maintain an ε-approximate quantile summary $Q(S)$ of the global frequency distribution S at the coordinator while minimizing the overall amount of communication between the coordinator and the remote sites. By providing a continuous ε-approximation guarantee, this quantile summary *Q(S)* at the coordinator can, at any time instant, be employed to answer:

- ε-approximate quantile-rank queries: given a domain value $v \in [U]$, we seek to find an approximate quantile rank $\bar{q}(v) \in [0,1]$ that is within ε of v's true quantile rank in *S*, i.e., find q(v) such that : $q(v) - \varepsilon \leq \bar{q}(v) \leq q(v) + \varepsilon$
- ε-approximate quantile-value queries: given a quantile rank $q \in [0,1]$, we seek a value $v = v(q) \in [U]$ whose quantile rank in *S* is within of *q*, i.e., $v = v(q)$ so $q - \varepsilon \leq q(v) \leq q + \varepsilon$.

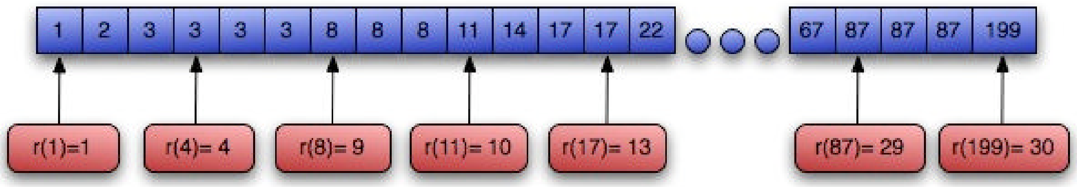

Figure 2.3: Example of quantile rank

The Cormode et al. notion of approximate quantile summaries for *s* is identical to that of all research in approximate quantiles [15][16][17][18].

An approximate quantile-value query is essentially the dual of an approximate quantile-rank query, and can be easily answered with *O(logU)* quantile-rank queries using binary research to determine a value *v* that generates an approximate quantile rank in the desiderate range *[q-ε, p+ε]* for the original query *q* symmetrically , we could answer approximate quantile-rank queries with a bounded number of approximate quantile-value queries.

2.3.3 Cormode et al. quantile tracking solution

Cormode et al. quantile-tracking scheme [7] is based in each individual remote site *j* continuously monitoring the quantile of its local update stream $S_j(j = 1,\ldots,k)$. When a certain amount of change is observed locally, then the site may communicate with the coordinator in order to update the coordinator with more recent information about its local update stream and, then, resumes monitor its local updates. The goal of this solution is to ensure strong ε-approximation guarantees for quantile queries over $S = U_j\ S_j$ at the coordinator while minimizing the amount of communication with remote sites. There are important design data that this solution should strive for:

- Summary-based Information Exchange. Rather than shipping the complete frequency distribution for their local streams S_j to the coordinator, remote sites only communicate concise quantile summaries $Q(S_j)$ of their locally-observed updates. The size of the $Q(S_j)$ summary depends critically on the desired ε-approximation guarantees at the coordinator site.
- Stability. Intuitively, the stability property means that, provided the local distributions at remote sites remain approximately the same, there is no need for communication between the remote sites and the coordinator. The interpretation of this property depend on the ability to model the similarity of the up-to-date local distribution S_j to their past behavior. As long as the models accurately capture the true behavior of the local update streams, no communication between the remote site and the coordinator is necessary.
- Minimal global Information Exchanges. Even though remote sites communicate only summary information on their local streams, as the number of site k increases, the communication penalty for interrogating all remote sites becomes inordinately high. Hence, the aim of this model to avoid solutions that may require regular collection or broadcasting of information from/to every remote site in the system. For instance, a scheme that distributes information on the global quantiles over S to all remote sites would typically need to rebroadcast up-to-date global-quantile information to sites (either periodically or during some “global resolution” stage [11]) in order to ensure correctness.

The solution is based on remote sites continuously monitoring local constrains on the quantile distributions or their local update streams S_j, and contacting the coordinator with an appropriate quantile summary $Q(S_j)$ once these local quantile constraints are violated. Briefly, the tracking scheme splits the allowed error tolerance ε at the coordinator into two distinct components ϕ and θ; that is, $\varepsilon = \phi+\theta$, where:

- ϕ captures the error of local quantile summaries communicated to the coordinator.

- θ captures the deviation of local quantiles at each remote site based on locally observed updates since the last communication with the coordinator.

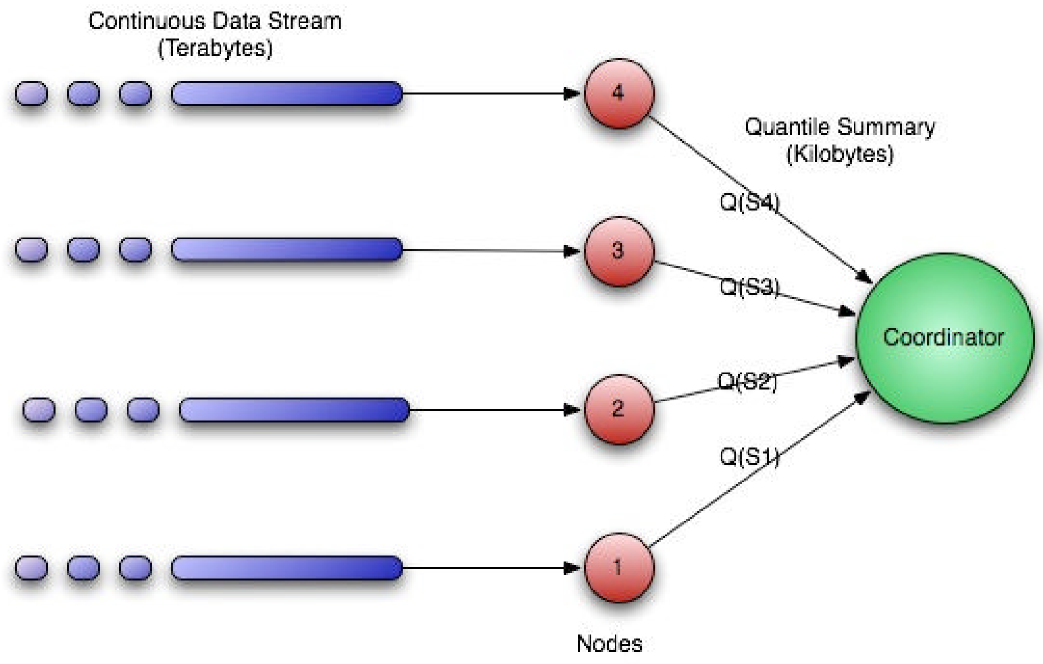

Figure 2.4: Size of streams

Thus a local quantile summary $Q(S_j)$ last communicated to the coordinator at time t carries an approximation error in the order of ϕ with respect to the snapshot of local stream S_j at time t, whereas θ bunds the deviation of local quantiles with respect to the snapshot-summary information sent to the coordinator. A larger θ value allows for larger local deviations since the last communication and, therefore, implies fewer communications to the coordinator but, since $\varepsilon = \theta+\phi$, for a given tolerance ε, the size of ϕ-approximate summary $Q(S_j)$ sent during each communication is larger. Each local quantile summary $Q(S_j)$ communicated to the coordinator at time t gives a picture of the snapshot of the S_j stream at time t. In order to achieve the stability property, a crucial component of this solution is the concept of concise prediction models that may be communicated from remote sites to the coordinator in attempt to accurately capture the anticipated behavior of local streams. The idea is that the coordinator employs the prediction model for site j to predict the current state of the S_j stream when estimating the global, up-to-date quantiles for S, remote site j employs the same prediction model to check for the deviation of its local quantiles with respect to the corresponding predictions at the coordinator. Thus, as long as the prediction model accurately capture the local update behavior at remote sites, no communication is needed. Note that prediction model is local information for a specific remote site, and can be computed either by remote site itself or by the coordinator. Since the prediction models are also part of the information exchanged between the remote sites and the coordinator, it is crucial to keep them simple and concise.

2.3.4 Cormode et a. basic tracking scheme

Fix a remote site *j*, and let *Q(S$_j$)* denote the collection of ϕ-quantile values of its local update stream *S$_j$* ; that is, *Q(S$_j$)* comprises $\left\lceil \frac{1}{\phi} \right\rceil + 1$ values $v_{0,j},...,v_{\left\lceil \frac{1}{\phi} \right\rceil,j}$ such that quantile rank of the i^th value $v_{i,j}$ in the local stream *S$_j$* is $q_j(v_{i,j}) = i\phi$ for *i* =0,1,…, $\left\lceil \frac{1}{\phi} \right\rceil^2$. In particular, note that $v_{0,j}$ and $v_{\left\lceil \frac{1}{\phi} \right\rceil,j}$ are the minimum and the maximum values observed in the stream *S$_j$*. It is not difficult to see that the above-described collections of ϕ-quantile values $Q(S_j) = \{v_{i,j} : i = 0,..., \left\lceil \frac{1}{\phi} \right\rceil\}$ is a $\frac{\phi}{2}$-approximate quantile summary for stream *S$_j$*. In this basic tracking scheme, remote sites can communicate their ϕ-quantile values summary $Q(S_j) = \{v_{i,j} : i = 0,..., \left\lceil \frac{1}{\phi} \right\rceil\}$ along with a concise prediction model for their local updates to the coordinator site. Let S_j denote the snapshot of the local stream last communicated, through $Q(S_j)$ to the coordinator, and let $N_j = |Q(S_j)|$; also let n_j denote the total number of element updates to the S_j multi-set since the last communication. Thus, the size of up-to-date local stream at site *j*, denoted by $S_j \cup \Delta S_j$, is $N_j + n_j$, while the size of the up-to-date local stream $S = \cup_j (S_j \cup \Delta S_j)$ is

$N = |S| = \sum_j N_j + n_j$. Obviously, when site j communicates with the coordinator, it sets $N_j \leftarrow N_j + n_j$ and reset $n_j \leftarrow 0$.

After shipping $Q(S_j) = \{v_{i,j} : i = 0,..., \left\lceil \frac{1}{\phi} \right\rceil\}$ and a corresponding prediction model to the coordinator, the site j continuously monitors the state of its local quantile values $v_{i,j}$ in its up-to-date stream. More specifically, for each local quantile value $v_{i,j}$, site j monitors both its true absolute rank $r_j(v_{i,j})$ in $S_j \cup \Delta S_j$, as well as its predicted absolute rank $r_j^p(v_{i,j})$ based on the prediction model communicated to the coordinator. Clearly, the exact methodology for computing the predicted rank $r_j^p(v_{i,j})$ depends on the specific prediction model being used.

In this solution a communication with the coordinator is triggered at site j only if, for some $v_{i,j}$, $|r_j^p(v_{i,j}) - r_j(v_{i,j})| > \phi(N_j + n_j)$; that is, the predicted true rank monitored quantile value in $S_j \cup \Delta S_j$ deviate more than $\phi(N_j + n_j)$. This condition is sufficient to provide strong ε-approximation guarantees for rank and quantile estimates based in the quantile summaries $Q(S_j)$.

Let $Q(S) = \cup_j Q(S_j) = \cup_j \{v_{i,j} : i = 0,..., \left\lceil \frac{1}{\phi} \right\rceil\}$ the global quantile summary used to approximate query answering at the coordinator is essentially a combination of $Q(S)$ and the per-site prediction models. More specifically, define $\hat{N} = \sum_j r_j^p(v_{\left\lceil \frac{1}{\phi} \right\rceil, j})$ that is, the sum of predicted maximum-element ranks across all S_j streams. Give a query value $v \in [U]$, the coordinator determines, for each site summary $Q(S_j)$, the index $i' = \arg\max_i \{v_{i,j} \in Q(S_j) : v_{i,j} < v\}$, and defines the bounding quantile value for v as $v_{i',j}$. It then estimates the absolute and quantile rank of v using the formulas:

- $\hat{r}(v) = \sum_j \frac{r_j^p(v_{i',j}) - r_j^p(v_{i'+1,j})}{2}$
- $\hat{q}(v) = \frac{\hat{r}(v)}{\hat{N}}$

Assume that, for each remote site *j* and local quantile value $v_{i,j} \in Q(S_j)$, we have $\left|r_j^p(v_{i,j}) - r_j(v_{i,j})\right| \le \phi(N_j + n_j)$ and $(r_j^p(v_{i',j}) - r_j^p(v_{i'+1,j})) \le 2\phi(N_j + n_j)$. Then for any value *v* ∈ *[U]*, the absolute- rank estimate $\hat{r}(v)$ at the coordinator site is a (θ+ϕ)-approximation to *v*'s true absolute rank *r(v)* in *S*; that is.

- $r(v) - (\phi + \theta)N \le \hat{r}(v) \le r(v) + (\phi + \theta)N$

2.3.5 Cormode et al. Prediction Models

Cormode et al. presented in their paper three different prediction models:

- Zero-information
- Synchronous-updates
- Update-rates

The most interesting model is the last; this model make the assumption of the presence of a global time , that the update are observer at each site *j* at a uniform local rate denoted by δ. This rate δ completely specifies the prediction model for site *j* and is exchanged between the coordinator and the site when communication takes place. The specific method of estimating the update rate has no effect on the correctness of the tracking, but a good estimates are important for reducing communication costs. For instance, δ_j can be defined either as a historical average over the entire history of updates at site *j*, or, more naturally, as an average update rate over a recent window of observed update behavior at the site. This model assumes that the quantile ranks of the values $v_{i,j}$ in the $Q(S_j)$ snapshot summaries last sent to the coordinator remain the same. Letting t_j denote the number of time steps since the last communication between the coordinator and the site *j*; the predicted ranks $r_j^p(v_{i,j})$ are defined as $r_j^p(v_{i,j}) = i\phi(N_j + \delta_j t_j)$ for each $i = 0, \ldots, \left\lceil \frac{1}{\phi} \right\rceil$. If the distributions at each site remain approximately the same, and the local-update rates are reasonably stable the update-rates model achieves stability making site-coordinator exchanges unnecessary in long run.

2.4 Non-linear update-rates

The Cormode et al. method works great under two principal conditions:

- Stability in the distribution of values at each site.
- Stability in the local update rate

In particular, if the distribution of values is not stable the rank of quantiles change a lot from one observation to another, and if the local update-rates are not stable the simple model $(N_j + n_j)$ does not work. The simplest solution for the first condition is to consider very long period, in this case the distribution will be stable in long run. For the second condition the simplest method is to fix update-rates e.g., one observation every $\frac{1}{10}$ of the global time period, so it can be managed by the δ_j value.

But in the real word the measurable things do not respect this two condition in most of cases. Take a router for example, is useless to use a very long period for the continuous tracking because usually the interest is focused to short-term analysis of the connections for detecting possible attacks such as Distributed Denial Of Service (DDOS), and also is impossible to have a stable update-rates, because is dependent from the number of connection in the fixed period.

For applying the quantile tracking solution to an environment with a non stable distribution of values and a no-constant update rate the first thing to do is to fix a global time and a period length. Consider $N_{j,t}$ the size of the stream observed at site j at the period t. The length of the period influence very much the tracking as we can see in the next chapter. The basic idea to extend the Cormode et al algorithm is to:

- predict θ-approximate $N_{j,t}$ for each site both at the coordinator site and in every node.
- let $\hat{N}_{j,t}$ denote the θ-approximated size of the stream at site j at period t; the node can calculate $\hat{N}_{j,t}$, and send the updated value to the coordinator if $\left|N_{j,t} - \hat{N}_{j,t}\right| \geq \theta N_{j,t}$, in this way an θ-approximate $\hat{N}_{j,t}$ is guarantee at the coordinator site.
- Use different way to compute the approximate predict rank values for a more precise values

In this way the success of the algorithm is not dependent by the frequency of the data, or by the kind of data; but only by how much is good the approximation algorithm for the computing of $\hat{N}_{j,t}$.

3. Forecasting Models

3.1 Time Series

The definition of Time Series is:
An ordered sequence of values of a variable at equally spaced time intervals [20].
The usage of time series models is twofold:

- Obtain an understanding of the underlying forces and structure that produced the observed data.
- Fit a model and proceed to forecasting, monitoring or even feedback and feedforward control.

Both of these goals require that the pattern of observed time series data is identified and more or less formally described. Once the pattern is established, we can interpret and integrate it with other. Regardless of the depth of our understanding and the validity of our interpretation of the phenomenon, we can extrapolate the identified pattern to predict future events.
Time Series Analysis is used for many applications such as: economic forecasting, sales forecasting, budgetary analysis and process and quality control [19].
According with this definition the size of the stream at site *j* at time *t*, $N_{j,t}$, is a data point of the time series $M_j = \left\{N_{j,1},...,N_{j,k} \mid k = (1,...,t)\right\}$.

3.1.1 Two General Aspects of Time Series Patterns

Most time series patterns can be described in terms of two basic classes of components: trend and seasonality. The former represents a general systematic linear or nonlinear component that changes over time and does not repeat or at least does not repeat within the time range captured by our data, e.g., a plateau followed by a period of exponential growth. The latter may have a formally similar nature, e.g., a plateau followed by a period of exponential growth, however, it repeats itself in systematic intervals over time. Those two general classes of time series components may coexist in real-life data. For example, sales of a company can rapidly grow over years but they still follow consistent seasonal patterns, e.g., as much as 25% of yearly sales each year are made in December, whereas only 4% in August.

	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1969
Jan	112	115	145	171	196	204	242	284	315	340	360	417
Feb	118	126	150	180	196	188	233	277	301	318	342	391
Mar	132	141	178	193	236	235	267	317	356	362	406	419
Apr	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125	172	183	229	234	270	318	355	363	420	472
Jun	135	149	178	218	243	264	315	374	422	435	472	535
Jul	148	170	199	230	264	302	364	413	465	491	548	622
Ago	148	170	199	242	272	293	347	405	467	505	559	606
Sep	136	158	184	209	237	259	312	355	404	404	463	508
Oct	119	133	162	191	211	229	274	306	347	359	407	461
Nov	104	114	146	172	180	203	237	271	305	310	362	390
Dec	118	140	166	194	201	229	278	306	336	337	405	432

Table 3.1: Series G dataset

This general pattern is well illustrated in a "classic" Series G data set [22] representing monthly international airline passenger totals (measured in thousands) in twelve consecutive years from 1949 to 1960, see table 3.1 and figure 3.1. In the plot of the Series G, shown in Figure 3.1, a linear trend emerges, indicating that the airline industry enjoyed a steady growth over the years, approximately four times more passengers traveled in 1960 than in 1949. At the same time, the monthly figures will follow an almost identical pattern each year, e.g., more people travel during holidays then during any other time of the year.

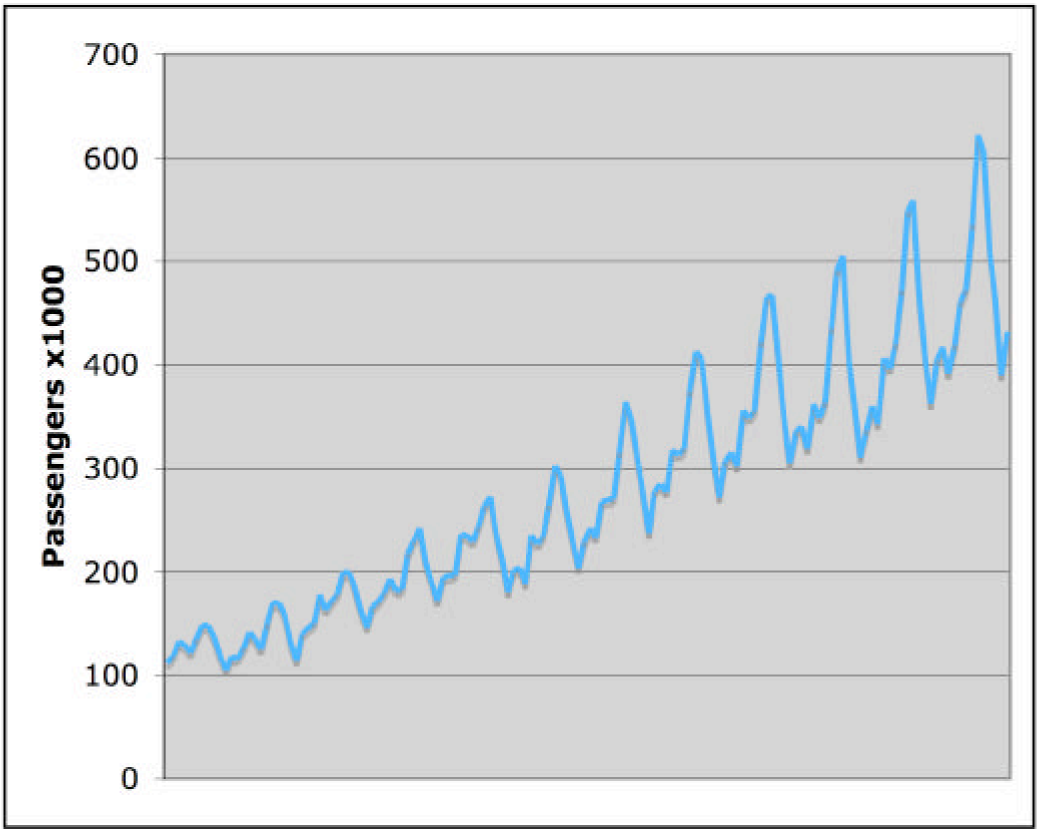

Figure 3.1: Series G data set

This example data file also illustrates a very common general type of pattern in time series data, where the amplitude of the seasonal changes increases with the overall trend, i.e., the variance is correlated with the mean over the segments of the series. This pattern which is called multiplicative seasonality indicates that the relative amplitude of seasonal changes is constant over time, thus it is related to the trend [19].

3.1.2 Trend Analysis

There are no proven "automatic" techniques to identify trend components in the time series data; however, as long as the trend is monotonous (consistently increasing or decreasing) that part of data analysis is typically not very difficult. If the time series data contain considerable error, then the first step in the process of trend identification is smoothing.

Smoothing always involves some form of local averaging of data such that the nonsystematic components of individual observations cancel each other out. The most common technique is moving average smoothing which replaces each element of the series by either the simple or weighted average of *n* surrounding elements, where *n* is the width of the smoothing "window" [21][23]. Medians can be used instead of means. The main advantage of median as compared to moving average smoothing is that its results are less biased by outliers, within the smoothing window. Thus, if there are outliers in the data, e.g., due to measurement errors, median smoothing typically produces smoother or at least more reliable curves than moving average based on the same window width. The main disadvantage of median smoothing is that in the absence of clear outliers it may produce more jagged curves than moving average and it does not allow for weighting.

In the relatively less common cases, when the measurement error is very large, the distance weighted least squares smoothing or negative exponentially weighted smoothing techniques can be used. All those methods will filter out the noise and convert the data into a smooth curve that is relatively unbiased by outliers. Series with relatively few and systematically distributed points can be smoothed with bicubic splines.

Fitting a function. Many monotonous time series data can be adequately approximated by a linear function; if there is a clear monotonous nonlinear component, the data first need to be transformed to remove the nonlinearity. Usually a logarithmic, exponential, or polynomial function can be used.

3.1.3 Analysis of Seasonality

Seasonal dependency, seasonality, is another general component of the time series pattern. The concept was illustrated in the example of the airline passengers data above. It is formally defined as correlational dependency of order *k* between each i^{th} element of the series and the $(i-k)^{th}$ element [24] and measured by autocorrelation, i.e., a correlation between the two terms; *k* is usually called the lag. If the measurement error is not too large, seasonality can be visually identified in the series as a pattern that repeats every *k* elements.

Seasonal patterns of time series can be examined via correlograms. The correlogram displays graphically and numerically the autocorrelation function (ACF), that is, serial correlation coefficients, and their standard errors for consecutive lags in a specified range of lags, e.g., 1 through 30. Ranges of two standard errors for each lag are usually marked in correlograms but typically the size of auto correlation is of more interest than its reliability because we are usually interested only in very strong autocorrelations.

While examining correlograms one should keep in mind that autocorrelations for consecutive lags are formally dependent. Consider the following example. If the first element is closely related to the second, and the second to the third, then the first element must also be somewhat related to the third one, etc. This implies that the pattern of serial dependencies can change considerably after removing the first order auto correlation, i.e., after differencing the series with a lag of one.

Another useful method to examine serial dependencies is to examine the partial autocorrelation function (PACF); an extension of autocorrelation, where the dependence on the intermediate elements is removed. In other words the partial autocorrelation is similar to autocorrelation, except that when calculating it, the correlations with all the elements within the lag are partialled out [21][25]. If a lag of one is specified, i.e., there are no intermediate elements within the lag, then the partial autocorrelation is equivalent to auto correlation. In a sense, the partial autocorrelation provides a cleaner picture of serial dependencies for individual.

3.1.4 Removing serial dependency

Removing serial dependency. Serial dependency for a particular lag of *k* can be removed by differencing the series, that is converting each i^{th} element of the series into its difference from the $(i-k)^{th}$ element. There are two major reasons for such transformations.

First, one can identify the hidden nature of seasonal dependencies in the series. Remember that, as mentioned in the previous paragraph, autocorrelations for consecutive lags are interdependent. Therefore, removing some of the autocorrelations will change other auto correlations, that is, it may eliminate them or it may make some other seasonalities more apparent. The other reason for removing seasonal dependencies is to make the series stationary which is necessary for ARIMA and other techniques.

3.2 Choice of the forecasting model

The choice of the right forecasting model for the prediction of $\hat{N}_{j,t+1}$ is not simple because the accuracy of the model is strongly dependent from the nature of the time series. Make the best choice possible I use some real-world data-sets to evacuate the response of the algorithm with the data of a typical sensor network. I used one public dataset, the World Cup 1998 HTTP request data; obtained from the Internet Traffic Archive. Each request specified a timestamp, the server response code and the size of the object returned. In this data there are 26 differed servers, each one corresponding to a remote site, and the handles a varying number of requests, form few thousand to many million; the object sizes varied from few bites to several Megabyte size. The period utilized is from May 30th, to June 11th.
Time division is necessary in real world applications, because data are useless without time specifications, a global distribution frequency across all the life of the process is not so useful in continuous-monitoring queries, because the process usually tend to stability in the long period, and so the answers to the queries are the same also if the distribution of the process change for a period. If the distribution of the process changes for a certain time, this does not effect the global distribution until the length of the change is relevant in confront of the length of the process. For example if the system is monitoring a network with continuous-monitoring queries, if the network is on a DDOS attack , with an high probability the answers to the queries remain the same of before the attack, because the length of the distribution changes created by the attack is short if we compare it with the length of the whole process. The distribution stability is one of the property that permit to Cormode et al. tracking model to work in an efficient way, we can see a lot of update for the nodes to the coordinator when the system starts and after the system achieve the stability no updates are needed. With time steps the process is no stable because each step is a kind of new process, and with sliding window there is

the elimination of the elements and insertion of new elements that make the process intrinsic unstable. The time division of the process is the base of all the following analysis because it introduces instability inside the process. The figures 3.2 and 3.3 show two different types of time division of the dataset. In the first chart the data represented are the number of update for each hour of the day, i.e. from 1am to 2am and so on, each hourly update consist in hundreds of thousand of single updates at the nodes side, condensate in a single moment and distribution. In the second chart I use a sliding window with the length of sixty minute, every minute, the update of the minute observed is added to the window and the updates of the minute distant sixty from this is subtracted, each update consists in tens of thousand of single update at the nodes side. In the first case the length of the period *L* is twenty-four, as the hour of a day, in the second is sixty, because I make minute tracking. number of connection using a window size of sixty minutes. This graph is made with only twenty hours of analysis for the better show of the shape of the curve . The total update analyzed in the following analysis are more than seventeen thousand.

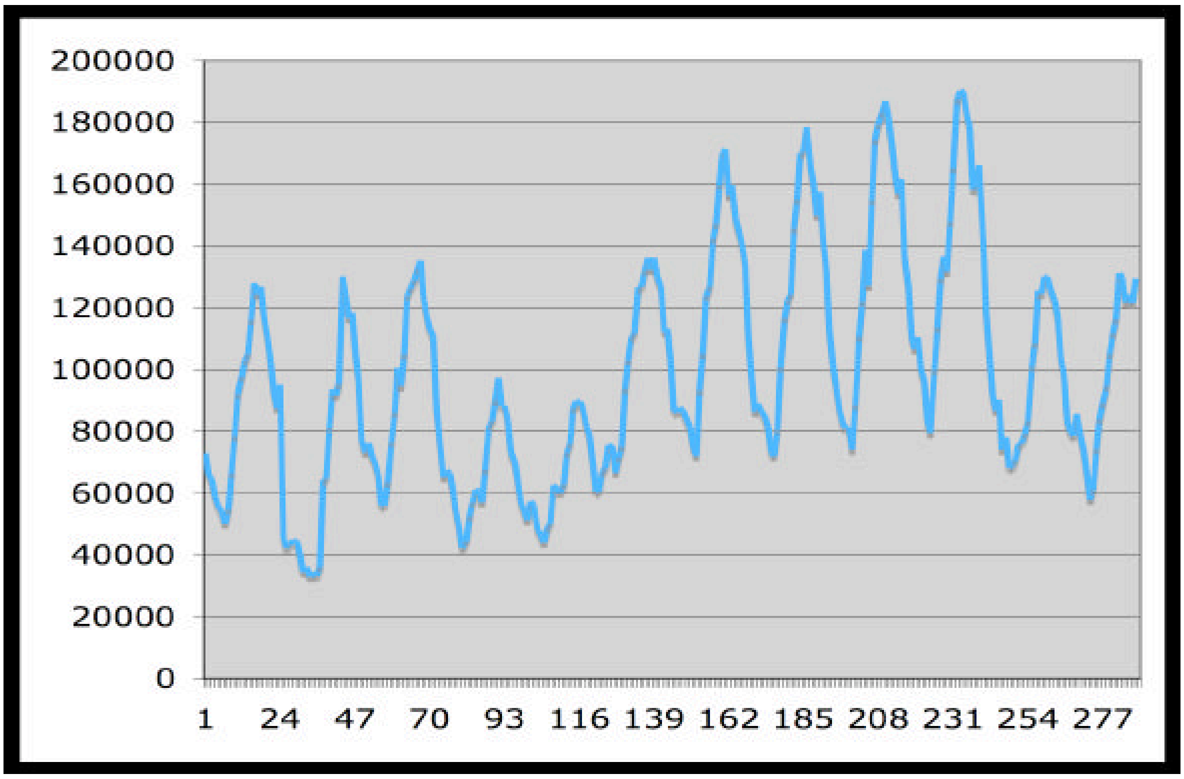

Figure 3.2: World Cup '98 HTTP requests: using time steps.

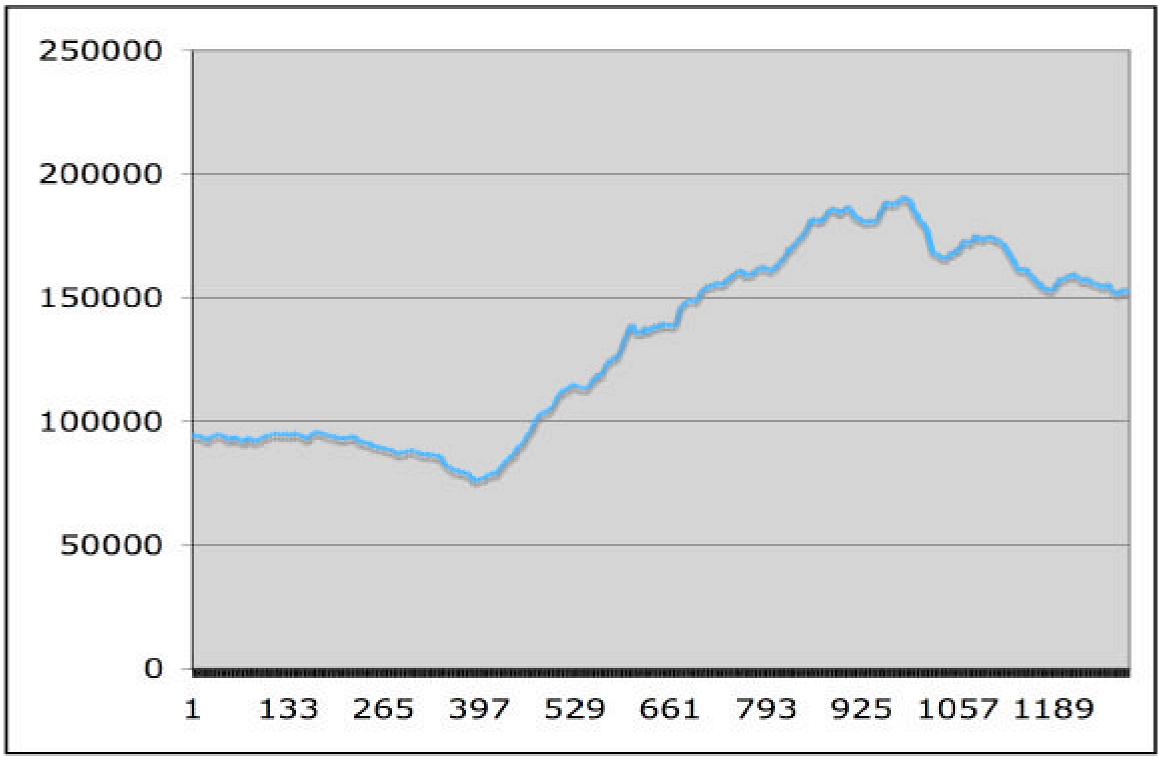

Figure 3.3: World Cup '98 HTTP requests: Sliding window

The main goal is so to forecast a complete new frequency distribution every new period of time, each frequency distribution contains hundreds thousand of values and can change a lot from the previous. First of all , we have to know with a good approximation the total number of updates at every node, so I analyzed two well known time series analysis algorithms.
I chose to analyze two forecasting models: ARIMA and Holt-Winter

3.2.1 ARIMA

Autoregressive Integrated Moving Average (ARIMA) model is a generalization of an autoregressive moving average or (ARMA) model. ARIMA models are, in theory, the most general class of models for forecasting a time series which can be stationarized by transformations such as differencing and logging. In fact, the easiest way to think of ARIMA models is as fine-tuned version of random-walk and random-trend models: the

fine-tuning consists of adding lags of the differenced series and/or lags of the forecast errors to the prediction equation, as needed to remove any last traces of autocorrelation from the forecast errors. Lags of the differenced series appearing in the forecasting equation are called "auto-regressive" terms, lags of the forecast errors are called "moving average" terms, and a time series which needs to be differenced to be made stationary is said to be an "integrated" version of a stationary series. Random-walk and random-trend models, autoregressive models, and exponential smoothing models (i.e., exponential weighted moving averages) are all special cases of ARIMA models [28].
A nonseasonal ARIMA model is classified as an "ARIMA(p,d,q)" model, where:

- p is the number of autoregressive terms,
- d is the number of nonseasonal differences, and
- q is the number of lagged forecast errors in the prediction equation.

To identify the appropriate ARIMA model for a time series, you begin by identifying the order(s) of differencing needing to stationarize the series and remove the gross features of seasonality, perhaps in conjunction with a variance-stabilizing transformation such as logging or deflating. If you stop at this point and predict that the differenced series is constant, you have merely fitted a random walk or random trend model. (Recall that the random walk model predicts the first difference of the series to be constant, the seasonal random walk model predicts the seasonal difference to be constant, and the seasonal random trend model predicts the first difference of the seasonal difference to be constant--usually zero). However, the best random walk or random trend model may still have autocorrelated errors, suggesting that additional factors of some kind are needed in the prediction equation.

- ARIMA(0,1,0) or random walk: there are two strategies for eliminating autocorrelation in forecast errors. One approach, which we first used in regression analysis, was the addition of lags of the stationarized series. For example, suppose we initially fit the random-walk-with-growth model to the time series Y. The prediction equation for this model can be written as:

  $$\hat{Y}_t - Y_{t-1} = \mu,$$

  where the constant term is the average difference in Y. This can be considered as a degenerate regression model in which DIFF(Y) is the dependent variable and there are no independent variables other than the constant term. Since it includes (only) a nonseasonal difference and a constant term, it is classified as an "ARIMA(0,1,0) model with constant." Of course, the random walk without growth would be just an ARIMA(0,1,0) model without constant

- ARIMA(1,1,0) or differenced first-order autoregressive model: If the errors of the random walk model are autocorrelated, perhaps the problem can be fixed by adding one lag of the dependent variable to the prediction equation--i.e., by regressing DIFF(Y) on itself lagged by one period.

  This would yield the following prediction equation:

  $$\hat{Y}_t - Y_{t-1} = \mu + \phi(Y_{t-1} - Y_{t-2})$$

  which can be rearranged to:

  $$\hat{Y}_t = Y_{t-1} + \mu + \phi(Y_{t-1} - Y_{t-2})$$

  This is a first-order autoregressive, or "AR(1)", model with one order of nonseasonal differencing and a constant term, i.e., an "ARIMA(1,1,0) model with constant." Here, the constant term is denoted by μ and the autoregressive coefficient is denoted by "phi", in keeping with the terminology for ARIMA models popularized by Box and Jenkins.

- ARIMA(0,1,1) without constant or simple exponential smoothing: Another strategy for correcting autocorrelated errors in a random walk model is suggested by the simple exponential smoothing model. Recall that for some nonstationary time series, the random walk model does not perform as well as a moving average of past values. In other words, rather than taking the most recent observation as the forecast of the next observation, it is better to use an average of the last few observations in order to filter out the noise and more accurately estimate the local mean. The simple exponential smoothing model uses an exponentially weighted moving average of past values to achieve this effect. The prediction equation for the simple exponential smoothing model can be written in a number of mathematically equivalent ways, one of which is: $\hat{Y}_t = Y_{t-1} + \mu - \theta e_{t-1}$

  where e_{t-1} denotes the error at period *t-1*). The coefficient of the lagged forecast error is denoted by the Greek letter θ and it is conventionally written with a negative sign for reasons of mathematical symmetry. "Theta" in this equation corresponds to the quantity $(1-\alpha)$ in the exponential smoothing formulas we studied earlier. When a lagged forecast error is included in the prediction equation as shown above, it is referred to as a "moving average" (MA) term. The simple exponential smoothing model is therefore a first-order moving average ("MA(1)") model with one order of nonseasonal differencing and no constant term, i.e., an "ARIMA(0,1,1) model without constant".

- ARIMA(1,1,1): The features of autoregressive and moving average models can be "mixed" in the same model. For example, an ARIMA(1,1,1) model with constant would have the prediction equation:

  $\hat{Y}_t = Y_{t-1} + \mu + \phi(Y_{t-1} - Y_{t-2}) - \theta e_{t-1}$.

  Normally, though, we will try to stick to "unmixed" models with either only-AR or only-MA terms, because including both kinds of terms in the same model sometimes leads to overfitting of the data and non-uniqueness of the coefficients.

The seasonal part of an ARIMA model has the same structure as the non-seasonal part: it may have an AR factor, an MA factor, and/or an order of differencing. In the seasonal part of the model, all of these factors operate across multiples of lag s, the number of periods in a season.

A seasonal ARIMA model is classified as an ARIMA(p,d,q)x(P,D,Q) model, where:

- P: is the number of seasonal autoregressive (SAR) terms.
- D: is the number of seasonal differences.
- Q: is the number of seasonal moving average (SMA) terms.

In identifying a seasonal model, the first step is to determine whether or not a seasonal difference is needed, in addition to or perhaps instead of a non-seasonal difference. You should look at time series plots and ACF and PACF plots for all possible combinations of 0 or 1 non-seasonal difference and 0 or 1 seasonal difference.

If the seasonal pattern is both strong and stable over time, e.g., high in the Summer and low in the Winter, or vice versa), then you probably should use a seasonal difference regardless of whether you use a non-seasonal difference, since this will prevent the seasonal pattern from "dying out" in the long-term forecasts.

The main problem is that ARIMA assumes that the time series is stationary; if the process is not stationary it would be differenced one or more time to achieve stationarity. For using ARIMA, is necessary to check at every update if the series is stationary and if is not is necessary to differencing it. This is too complex to do, and also require some communication between the node and the coordination for deciding the order of differencing.

3.2.2 Holt-Winter

Exponential smoothing has proven through the years to be very useful in many forecasting situations. It was first suggested by C.C. Holt in 1957 and was meant to be used for non-seasonal time series showing no trend. He later offered a procedure (1958) that does handle trends. Winters (1965) generalized the method to include seasonality, hence the name "Holt-Winters Method". Holt-Winters algorithm is basically a quantitative forecasting method that uses mathematical recursive functions to predict the trend behavior. It uses a time series model to make predictions assuming that the future will follow the same pattern as the past. In particular if we have seasonal patterns corresponding to weekly periodicity, this means that we should use a factor in our equations that uses information from past weeks to make a more accurate prediction of what’s going to happen in the future. The Holt-Winters algorithm equations [20] are given by:

- $S_t = \alpha \dfrac{y_t}{I_{t-L}} + (1-\alpha)(S_{t-1} - b_{t-1})$
- $b_t = \gamma(S_t - S_{t-1}) + (1-\gamma)b_{t-1}$
- $I_t = \beta \dfrac{y_t}{S_t} + (1-\beta)I_{t-L}$
- $F_{t+m} = (S_t + mb_t)I_{t-L+m}$

Where:

- y_t is the observed value at time t.
- S_t is the smoothed observation at time t.
- b_t is the trend smoothing at time t.

- I_t is the seasonal smoothing at time t. In the equations we can see that we always use *t-L* because we are using information from previous season.
- L is the number of periods that complete 1 season. If we are making our measurements every 3 minutes and the season is 1 week long, L= 3360, there are 10080 minutes in one week so just divide this number by 3.
- F_{t+m} is the forecast/prediction at m periods ahead
- m is the number of periods ahead we want to predict. If we are making our measurements every *n* minutes, an m=3 will means that we are predicting *3n* minutes in the future.
- α is the overall smoothing parameter.
- β is the seasonal smoothing parameter.
- γ is the trend smoothing parameter.

α is basically the short term parameter. A large value of α will give a large weight to measurements very near in the past, while a small value of α will give more weight to measurements further in the past.
β is the trend parameter. A big value of β will give more weight to the difference of the last smoothed observations; while a little value of β will use information further in the past.
γ is the seasonal parameter. A big value of γ will give a big weight to the present relation between the observation and the smoothed observation, and little values of γ will give more weight to past weeks relation between the observation and the smoothed observation.

In the equations there are values corresponding to inexistent periods of time, like the case of S_{t-1} or b_{t-1}. When t=0, there is any values for *S* or *b*, so the model needs initial values. The same happen for the seasonal index *I* when *t<L*.
Initial Values for the observation smoothing:

- $S_0 = 0$

Initial Values for the trend smoothing:

- $b_0 = \dfrac{(y_{L+1} + y_{L+2} + \ldots y_{L+L} - y_1 - y_2 - \ldots - y_L)}{L^2}$

This value of the trend factor *b* is mainly an average of the differences in the observations of the first two period divided by the length of the period.

Initial Values for the Seasonal smoothing:

- $I_{k-L} = \sum_{i=0}^{1} \frac{y_{k+iL}}{\sum_{j=k+iL}^{(i+1)L} y_j}$

For *k*=1,2,3,…,L-1,L

This value of the seasonal index is basically making an average of the observed values of every season and then making an average of the period m of every season divided by the average of the observed values for its season.

I chose to use the Holt-Winter model because is not dependent form the stationarity of the time-series, this is very important because I can apply this model to different environment without modify the dataset with the differences, but I can adapt the model only choosing different parameters.

3.3 Choosing the parameters

The model parameters need to be set and tuned for the model to work well. There is no single optimal set of values, even restricted to data for a single variable. This is due to the interplay between multiple parameters in the model.

For example, consider two observations in sequence, y_t and y_{t+1}. The intercept α, slope β, and seasonal γ coefficients all absorb some part of the difference between y_t and y_{t+1}during the exponential smoothing update. It is safe to assume some of the difference is noise, so updates to the coefficients need not account for all of the difference between y_t and y_{t+1}. The values of α, β, and γ determine the relative share of the difference assigned to a changing baseline, a changing linear trend, and a changing seasonal coefficient [27].

Here are some common sense guidelines for setting parameters:

- α: At least one of α, β, and γ should allow adaptation in a short time frame. As seasonal updates occur infrequently for each coefficient (once per cycle), and the goal of is to capture a slowly changing linear trend, the most logical choice is α . Use exponential smoothing weights to make an educated choice for α . The sum of the most recent *n* weights is $1-(1-\alpha)^n$ and of course the sum of all weights is 1 (ignoring initialization). These facts can be manipulated to choose α using the formula: $\alpha = 1 - \exp(\frac{\ln(1 - total\_weight\_as\%)}{\#\_of\_time\_po\mathrm{int}\,s})$

  For example, if one wants observations in the last 45 minutes to account for 95% of the weights, and observations occur at five minute intervals (nine time points), then the formula yields α= 0.28. This formula can be rearranged using simple algebra to compute either the total weights as a percentage or the number of time points (elapsed time). For example, if α= 0.1, then the most recent hour of observations at five minute intervals (12 time points) accounts for 75% of the baseline prediction.
- β: As the purpose is to capture a linear trend longer than one seasonal cycle, it is logical to choose such seasonal cycle does not account for a majority of the exponential smoothing weights. The formula discussed previously applies with β replacing α . For example, if the period of the cycle is one day at one observation every five minutes (288 per day), then setting = 0.0024 will guarantee that observations within the last day account for less than 50% of the smoothing weights.
- γ: The seasonal adaptation parameter can also be selected using exponential smoothing weights using a variation of the previous formula. Note this single parameter controls both seasonal coefficient and deviation adaptation, on the assumption that seasonal trend and variability evolve together over time at roughly the same rate.

There is another way for choosing the parameters, the choice of the best parameters is based on experiments, as described in DBCAP [30]. To find the parameters α, β, and γ the minimization error algorithm used in this paper is the Minimum Mean Square Error (MMSE), where the error is the difference between the forecast and the observed values at time t. The minimization equation is given by:

$$Min\left(\frac{1}{K}\sum(F_t - y_t)^2\right)$$

restrictions: $0 \leq \alpha,\beta,\gamma \leq 1$

where:

- K: is the number of the observation.
- F_t: is the forecasted value at time *t*.
- y_t: is the real value at time *t*.

It's obvious that this solution is affordable on with the use of some training dataset, longer datasets for testing produce better parameters, making the assumption that the parameters that best describe the training dataset will describe well also the future values.I found that this is not the better method for finding the optimums parameters for the forecasting, because I have not to minimize the sum of the error, but the wrong prediction, because is not interesting how far the prediction is from the real value if the forecasted value is wrong. A predicted value is wrong if the error between the forecasted value and the real value exceeded a fixed percentage because the node send an update to the coordinator site only if: $\left|N_{j,t} - \hat{N}_{j,t}\right| \geq \theta N_{j,t}$

So if the goal is to minimize the communication cost between node and coordinator, we have to minimize the wrong forecasted values.

The best way for finding the parameters of the model is to use an iterative method that minimizes the number of the prediction errors.

This is the pseudo code:

```
while $\alpha$ <1
    while $\beta$ <1
        while $\gamma$ < 1
            for t =0 until t < length of training dataset
                calculate the forecasted value
                if forecasted value is a $\theta$-approximation of the real value
                    true++
                else
                    false++
            if true > stored-value
```

```
                save new parameters
        $\gamma = \gamma + 0,01$
    $\beta = \beta + 0,01$
$\alpha = \alpha + 0,01$
```

I choose a step of 0,01 for the increment on the parameters values because the speed of this algorithm with this step is acceptable and the parameters in the worst case are far ±0.01 from the optimal parameters. I tried to use 0.1 and 0.001 as steps for the parameters calculation, but in the first case the error is not acceptable, and in the second case the gain in more right values of use a parameter nearer the optimal is lower than the loss in time caused by one billion's iteration that a 0.001 one step required .

All the chart and graphs are generated using the data produces by the sensor network simulator described in *Chapter 4*. All the measurements are relative at the coordinator site. The dataset used is the World Cup '98 [26] dataset, for each update I have considered the timestamps for the time division, and the size of the HTTP request for the tracking. In all the next figures the real data are indicated by the blue color and the forecasted by the red color.

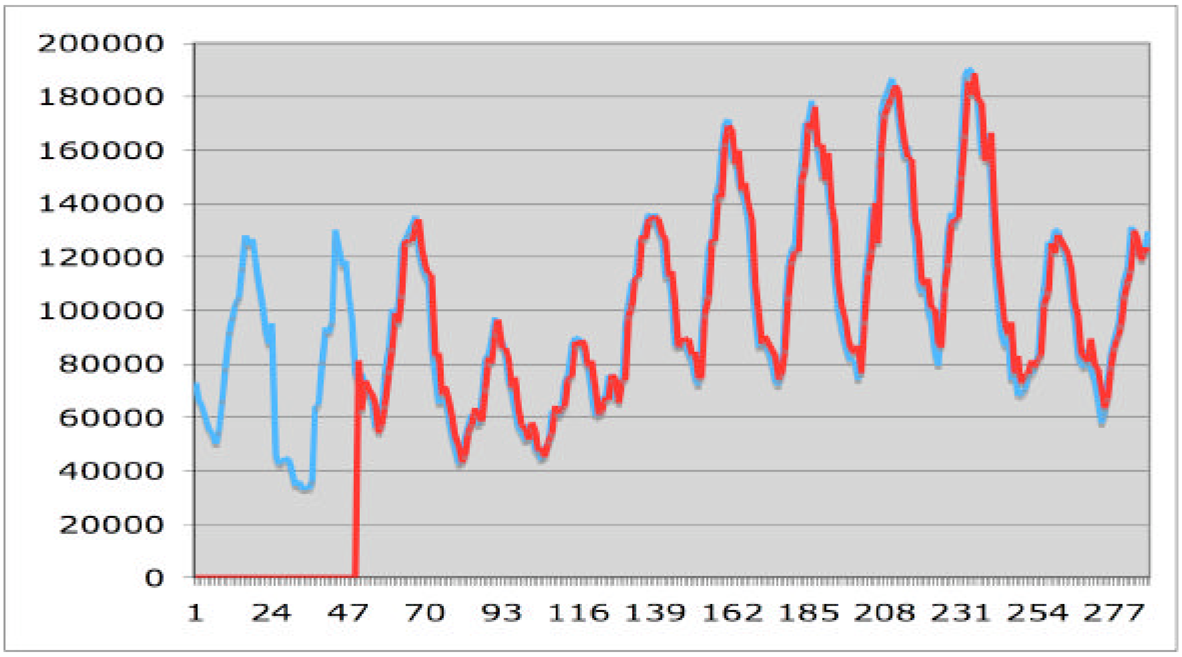

Figure 3.4: Data values, error permitted of 3%, using the MMSE. Time steps

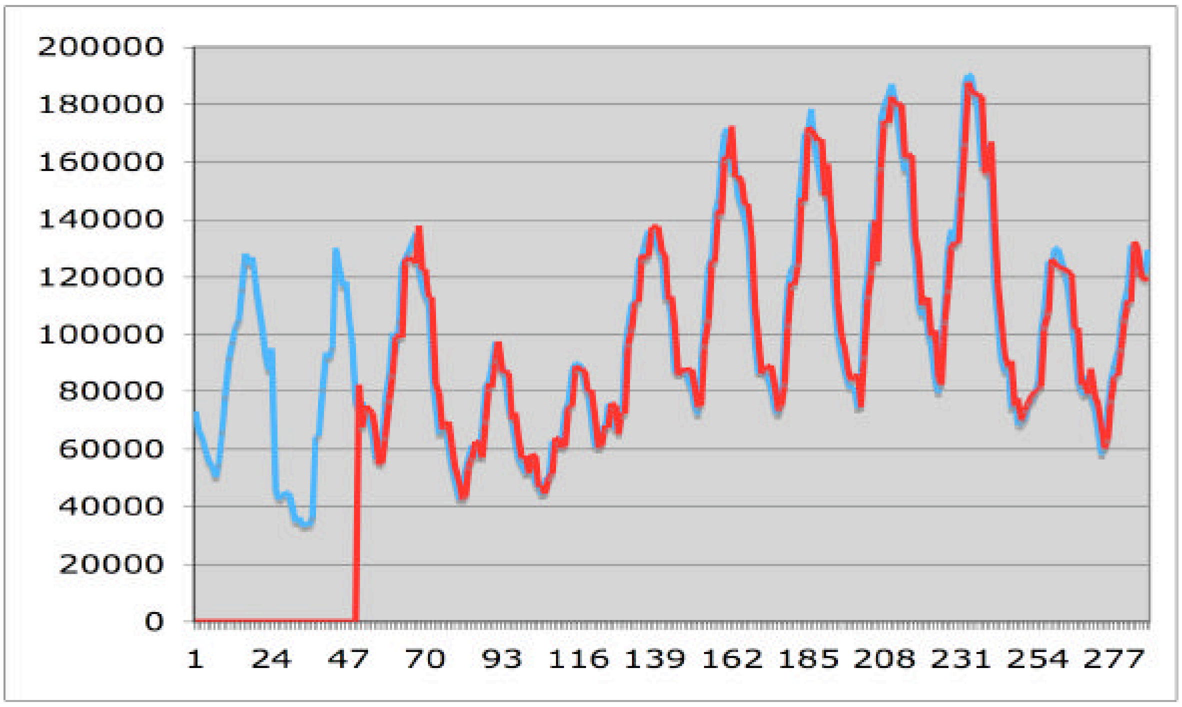

Figure 3.5: Data values, error permitted of 5%, using the MMSE. Time steps

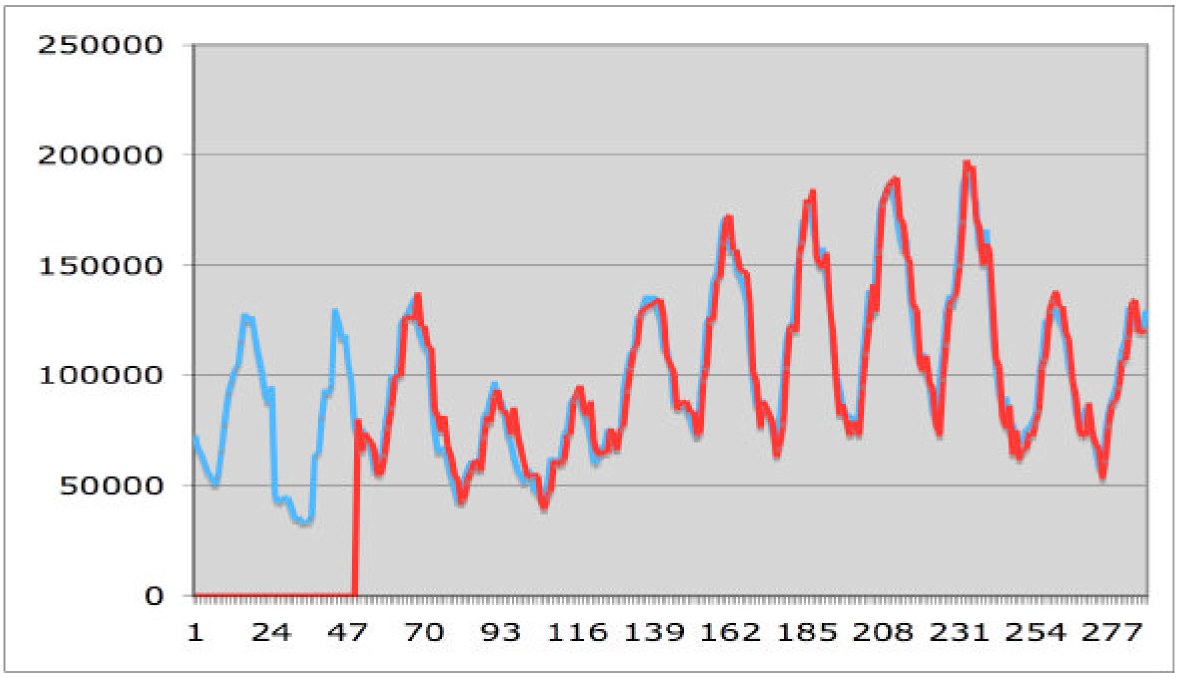

Figure 3.6: Data values, error permitted of 5%, using MWP. Time steps

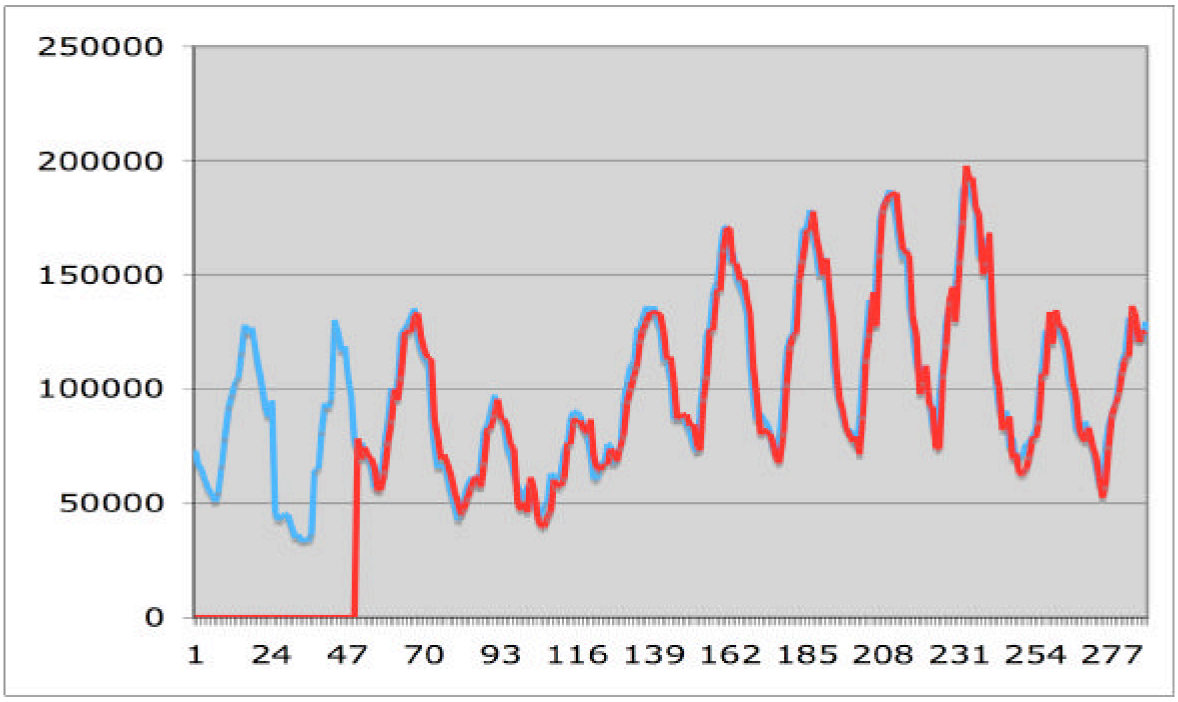

Figure 3.7: Data values, error permitted of 3%, using MWP. Time steps

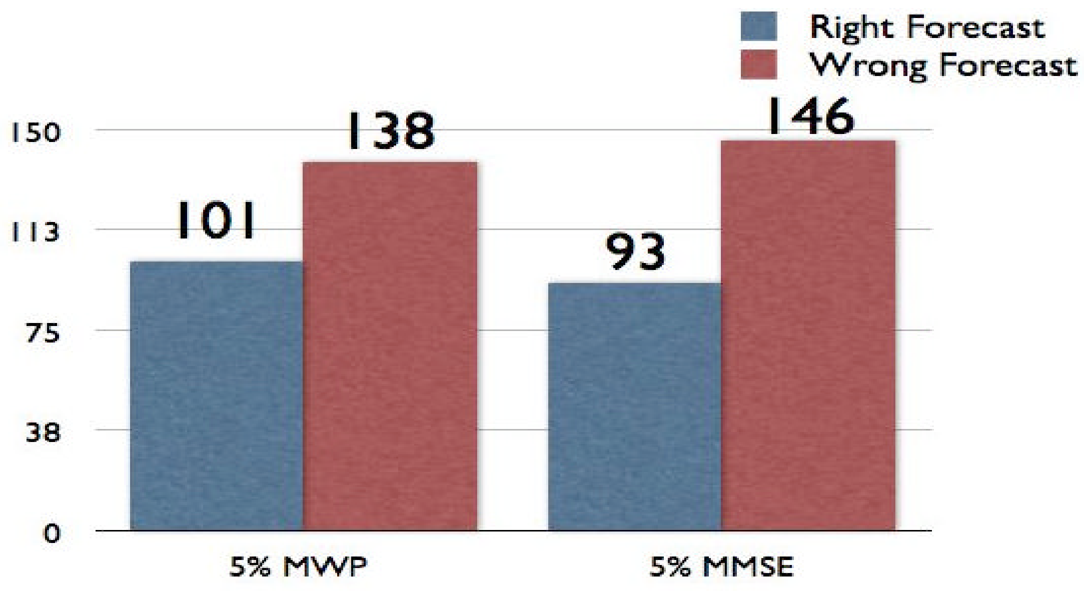

Figure 3.8: MPW vs. MMSE, error permitted 5%. Time steps

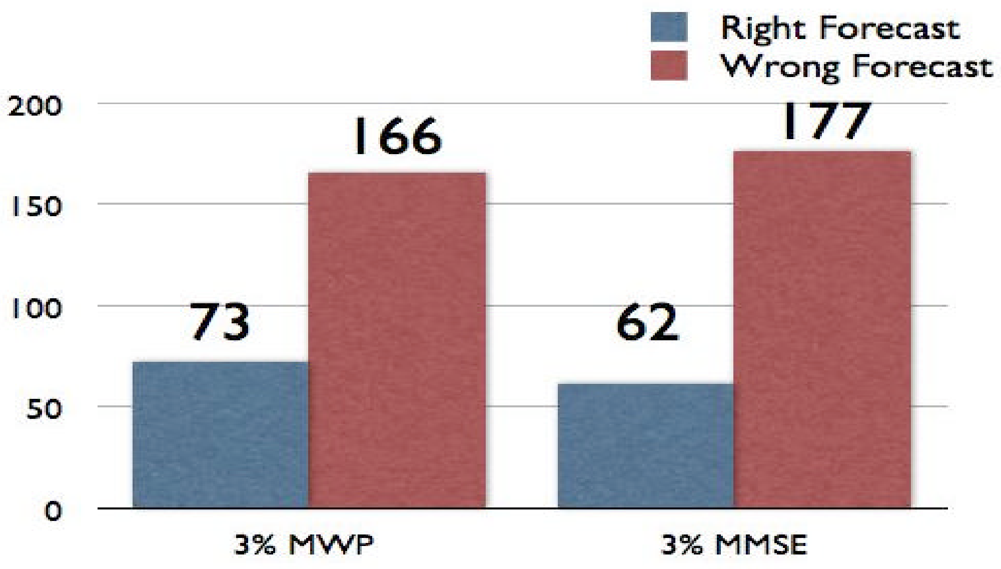

Figure 3.9: MPW vs. MMSE, error permitted 3%. Time steps

The first four figures above are graphs representing the real values and the forecasted values of *N* calculated at the coordinator site. The dataset is the World Cup '98 dataset [26], and *N* is collected every hour. The forecasted values start from the 48th because the first two L periods are used for the initial parameters of the Holt-Winter model. The last two figures show the increment in the performance determined by the use of a Minimum Wrong Prediction (MWP) algorithm, these data are extrapolated from the first four graphs, summarizing the situation at the end of the tests. In each case the parameters were calculated on a training dataset four periods L long, different form the testing dataset, but always from the Word Cup '98 dataset. The increment of performance using the MWP were of:

- 17% using θ=3%, 73 right prediction with the MWP against 62 using the MMSE
- 8% using θ=5%, 101 right prediction with the MWP against 93 using the MMSE

These data show that the performance of the MWP algorithm is better as low is the θ bound.

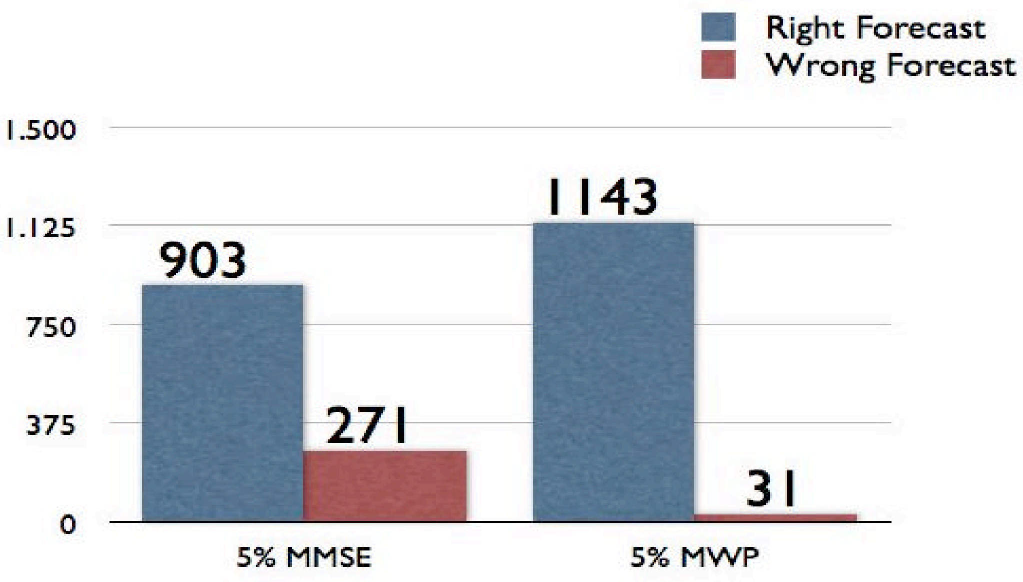

Figure 3.10: MPW vs. MMSE, error permitted 5%. Sliding Window

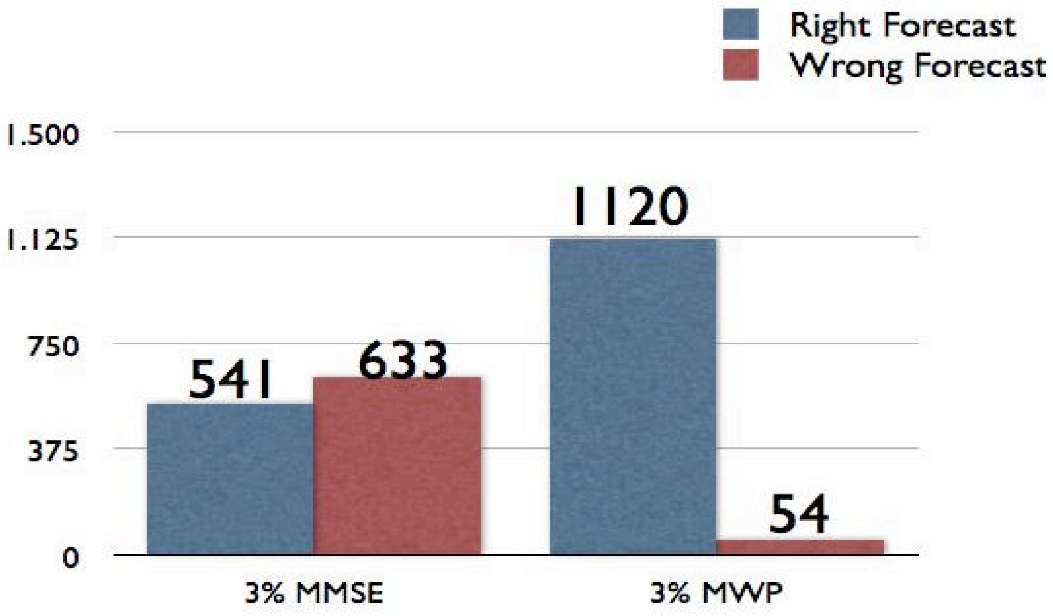

Figure 3.11: MPW vs. MMSE, error permitted 3%. Sliding Window

These two charts above represent the number of right and wrong forecast calculating using the holt-Winter algorithm, using two different techniques for the choice of the algorithms parameters. The graphs show that the Minimum Wrong Prediction permit to achieve good performance with the Holt-Winter, and show also that low θ using sliding window does not affect the quality of the forecast too much as happen using the time steps, maybe for the more stability of the process.

To summarize the performance of the MPW respect the MMSE using the siding window:

- 107% using θ=3%, 1120 right prediction with the MWP against 541 using the MMSE
- 26% using θ=5%, 101 right prediction with the MWP against 93 using the MMSE

New parameters are stored in the node memory, and are sent to the coordinator site with the training dataset, so the coordinator can start forecast values. In this way I can obtain the best parameters possible but the cost of computation is high because there are 100^3 cycle, and is dependent from the length of the training dataset.

If the distribution of the data changes a lot from the training set the calculated parameters con approximate well the new distribution, so when the model makes x consecutive error in the forecasting new parameters are needed. When this happen the node could use as training dataset the values measured in the past, and first calculates forecasted values from the previous parameters stored, then if the previous solution does not produce sufficient good forecasted values the node compute complete new parameters with the model above, but searching the new parameter around the old parameters for reducing the computational cost. If more than one algorithm of best parameters' research is executed in a short period of time the best thing to do is to recalculate the parameters starting from zero; because if the distribution change very much from the previous the parameters near the old are not the best possible, so for saving the communication cost is necessary increasing the computational cost.

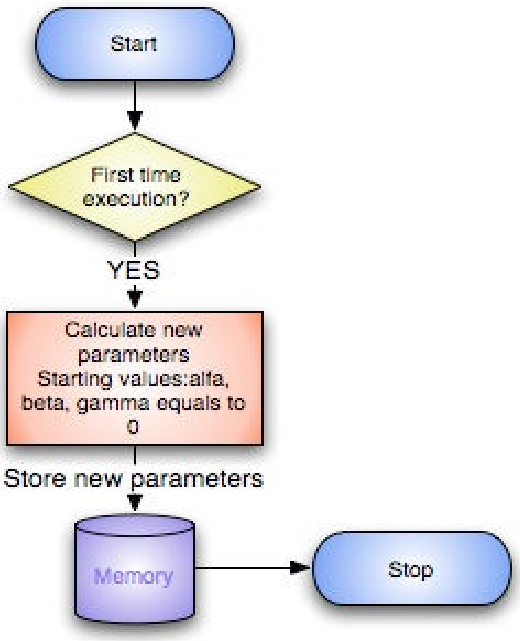
Start
First time execution?
YES
Calculate new parameters Starting values:alfa, beta, gamma equals to 0
Store new parameters
Memory
Stop

Figure 3.12: flow chart of the initial parameters calculation

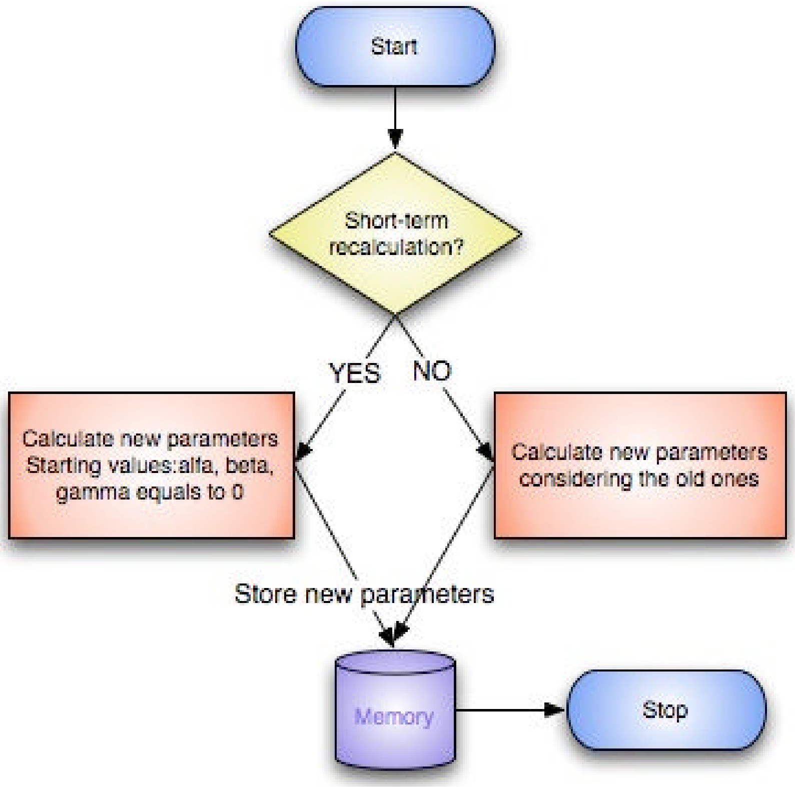
Start
Short-term recalculation?
YES
NO
Calculate new parameters Starting values:alfa, beta, gamma equals to 0
Calculate new parameters considering the old ones
Store new parameters
Memory
Stop

Figure 3.13: flow chart of the runtime parameters calculation

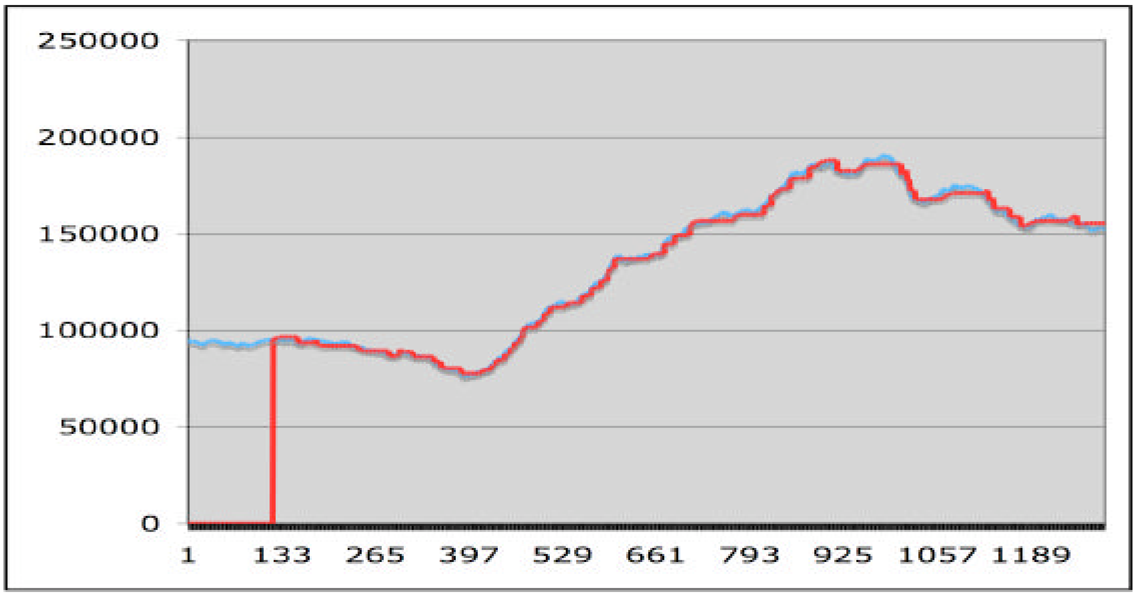

Figure 3.14: error permitted of 3%, using MWP, with the recalculation. Sliding window

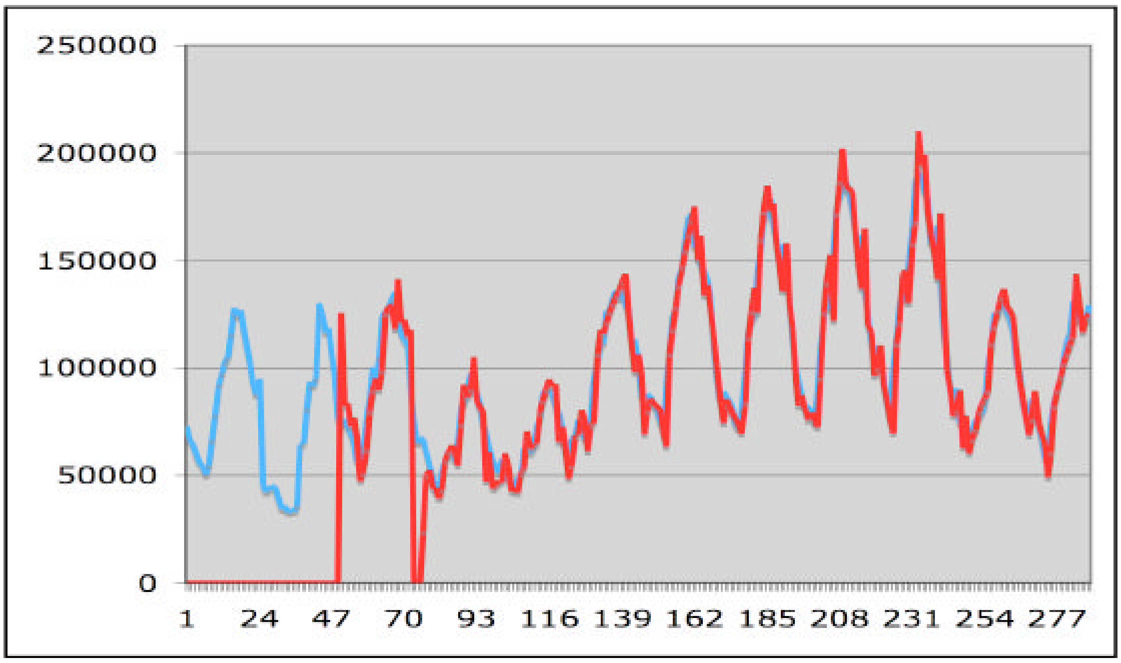

Figure 3.15: error permitted of 5%, using the MWP, with the recalculation. Time steps

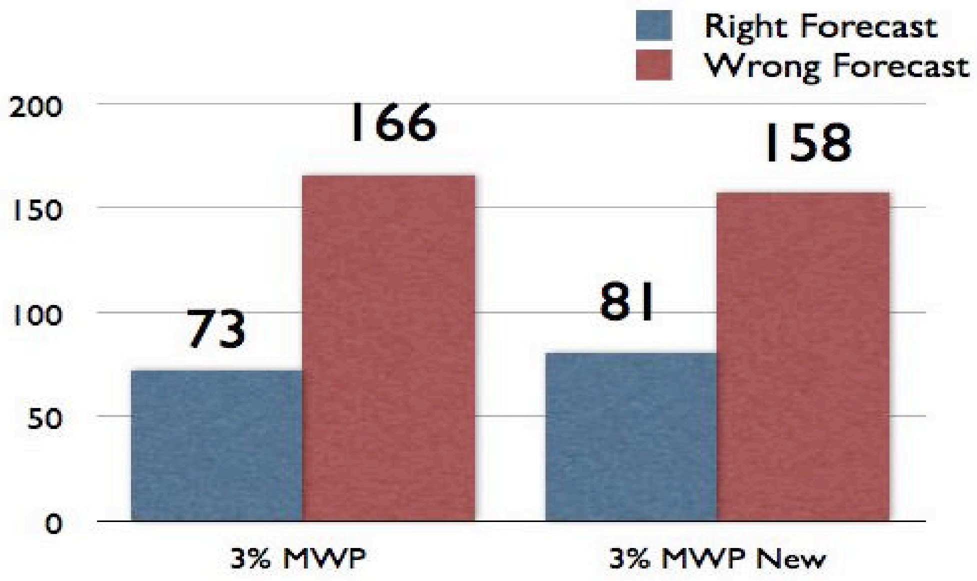

Figure 3.16: MPW vs. new MPW, error permitted 3%. Time steps

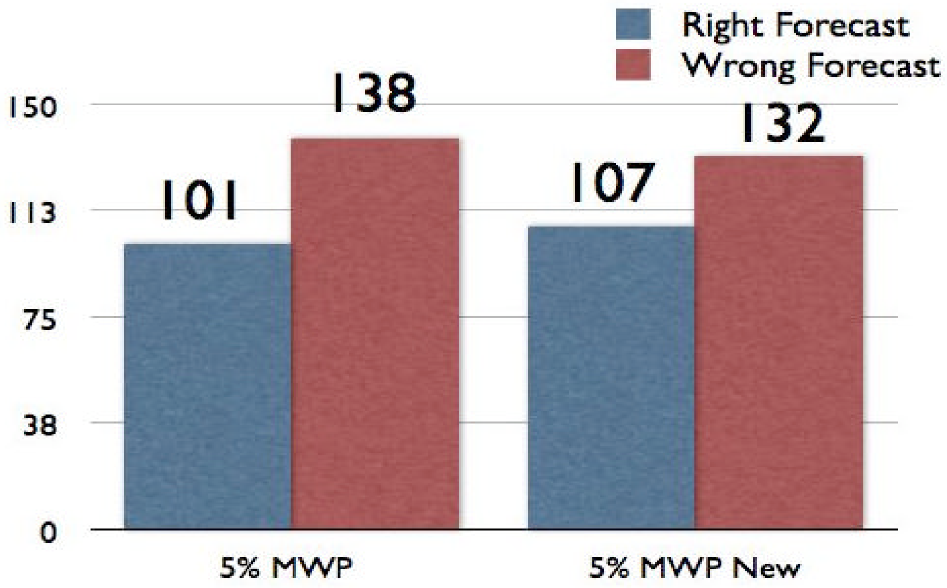

Figure 3.17: MPW vs. new MPW, error permitted 5%. Time steps

The two graphs, Figure 3.16 and Figure 3.17, above show the increment in the performance, using a system based on the time steps of one hour each, determined by the recalculation of the parameters if the distribution of update, at the node side, changes too much respect the previous distribution. In each case the initial parameters were calculated on a training dataset four periods L long, different form the testing dataset, but always from the Word Cup '98 dataset, and for the MWP algorithm this parameters did not change during the test, instead for the MWP algorithm the α, β, and γ values were recalculated after five consecutive wrong predictions, using as training data the past values observed. The increment of performance using the parameters recalculation was of:

- 10% using θ=3%, 81 right prediction with the MWP with recalculation against 73 using the old predictive model.
- 5.9% using θ=5%, 107 right prediction with the MWP with recalculation against 101 using the old predictive model.

Looking Figure 3.18 and Figure 3.19, representing the number of wrong and right forecasted values using the sliding window, we see that the increment of right values the difference is not high, because the performance of the normal MWP in a system using sliding window is already good, but considering only the wrong values decrease the new version of the MPW , the performance is very good.

- 12% less wrong predictions using θ=3%, 54 right prediction with the MWP with recalculation against 48 using the old predictive model
- 17% less wrong predictions using θ=5%, 31 right prediction with the MWP with recalculation against 26 using the old predictive model

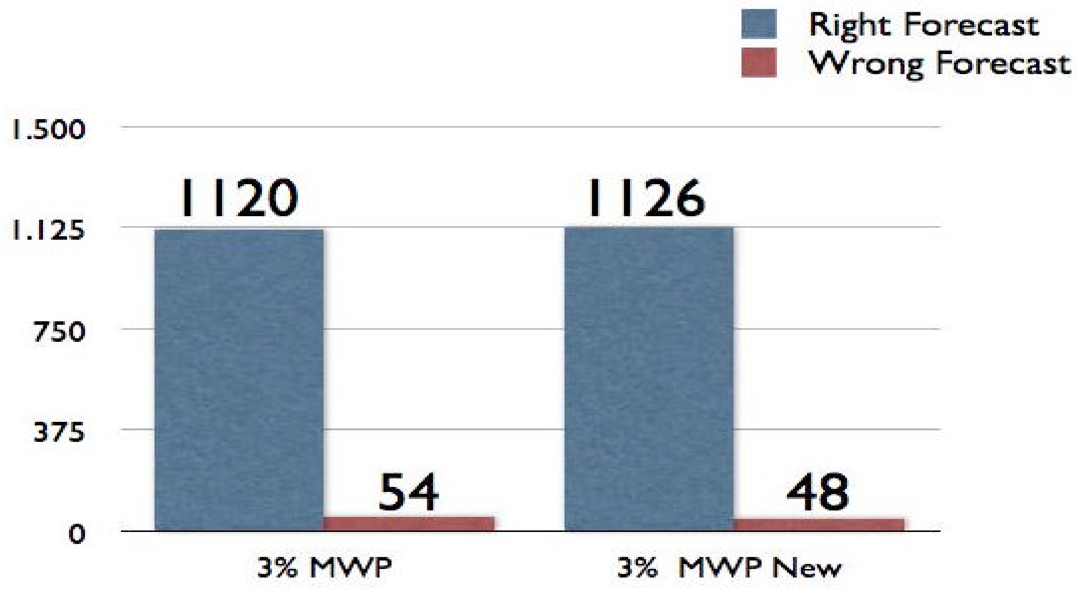

Figure 3.18: MPW vs. new MPW, error permitted 3%. Sliding window

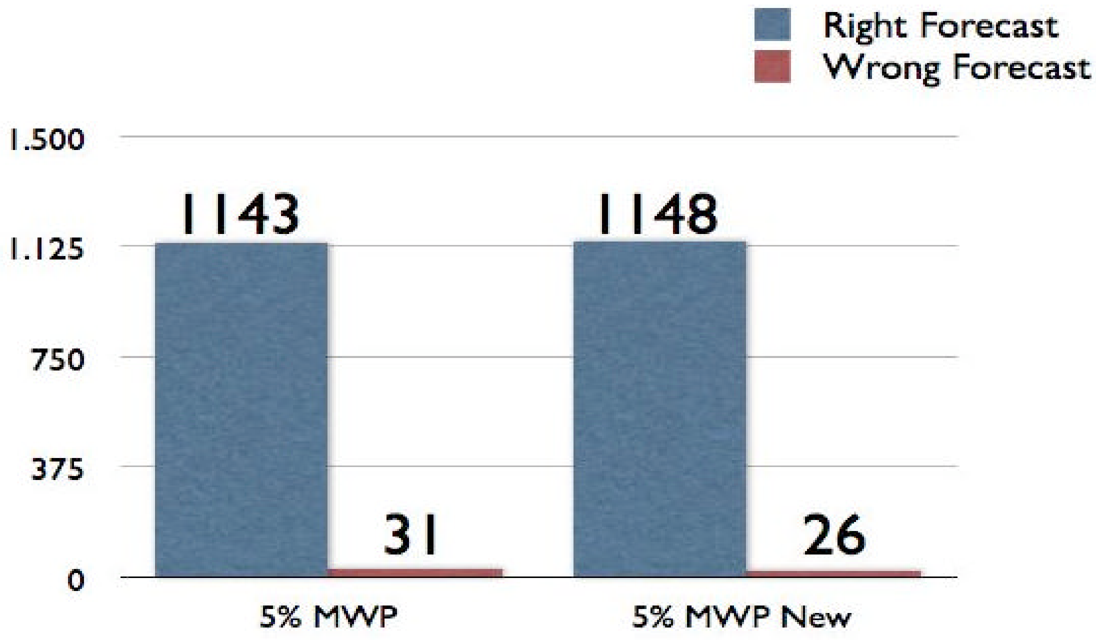

Figure 3.19: MPW vs. new MPW, error permitted 5%. Sliding window

3.4 Error Correction

The performances of this algorithm are good but not good enough for the optimization purpose that I want to achieve.

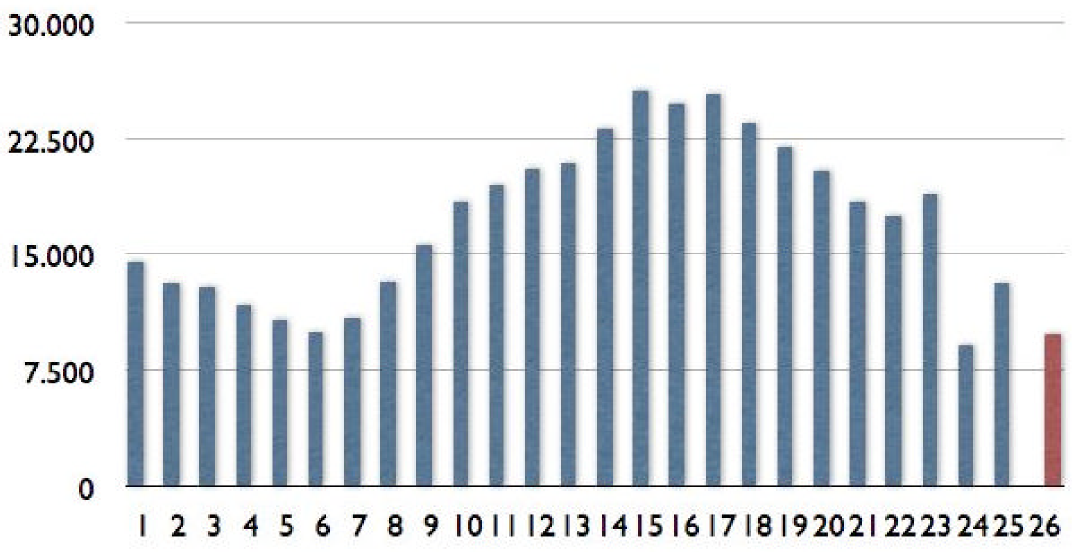

*Figure 3.20: Example of forecasted value (red) World Cup Data [**26**]*

In the Figure 3.20 is showed a graph representing the value of $N_{j,t}$, in blue, for t = *[1,25]*, t is long one hour in these measurements, the data are for one day and one hour. The red column represent the forecasted $\hat{N}_{j,t}$ for $t = 26$. This forecasted value is 9829. But the real value $N_{j,t}$ for t =26 is 12389, that is far than more 10 % from the forecasted value. My goal is to have a good prediction so more than 10% is not a good prediction, also because at the coordinator site the forecasted quantile rank values are (ϕ+θ)-approximate. So a high θ make also less precise the predictions of the quantile rank at the coordinator site. If the series shows seasonality and trend as happen typically in sensor networks data, I can do some consideration about the couples of sequential values distant L form each others. As we can se in the Figure 3.6 the difference from the values 1 and 2 is smaller than the difference between the values 25 and the forecast of 26. The first observation is that is obvious that the difference is different because the values are different, but I expect a smaller difference in the second couple because the values are lower. Is evident that for achieve better forecast I need to develop some adjustment for the Holt-Winter model.

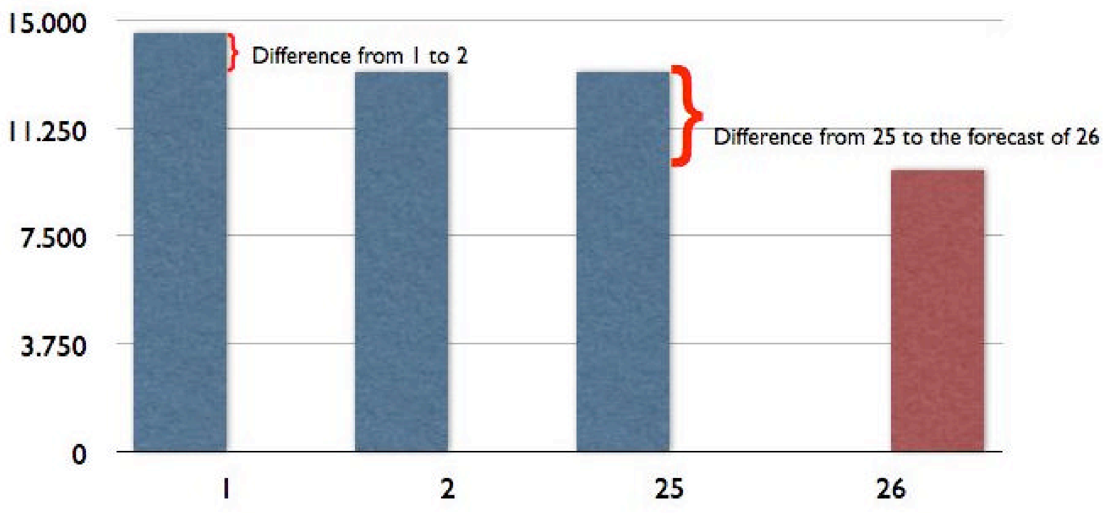

Figure 3.21: Two couples distant L compared

For achieve better forecasts I choose to insert an error correction term after the Holt-Winter model.

I assume that for every couple of sequential values of a time series, their difference divided by the average maintains a value similar to that calculated between the two values distant L from these.

So the formula for this value that have to be more or less constant between two couples distant L is:

$$\frac{(A_{i-1} - A_i)}{\frac{(A_{i-1} + A_i)}{2}} = c_i$$

where:

A_i: is the value of the time series at time i.

c_i: is the correction coefficient at time i.

The initial array of length L of correction coefficient is calculated with the expression above with 0<*i*≤L, so a training dataset is needed also for calculate the first c_i values. When the training is completed, after L times, the goal is to correct the ith forecasted value to make its value of c_i most similar, almost equal, as possible to the value of c_{i-L}.

Now I compare the expression of c_i, for i>L, with c_{i-L}:

$$\frac{\{[A_{i-1}-(B_i+\lambda_i)]\}}{\frac{[A_{i-1}+(B_i+\lambda_i)]}{2}}=c_{i-L}$$

I substitute to A_i the value $(B_i+\lambda_i)$, where:

B_i: is the forecasted value at time i with the Holt-Winter algorithm

λ_i: represent the distance of the forecasted value from the real one.

The only thing that we don't now inside the above expression is λ_i, so we can explicitize the expression above and:

$$\frac{[(2-c_{i-L})A_{i-1}+(-2-c_{i-L})B_i]}{2+c_{i-L}}=\lambda_i$$

λ_i is calculated with the correction coefficient at time i-L, and its used to correct the forecasting .

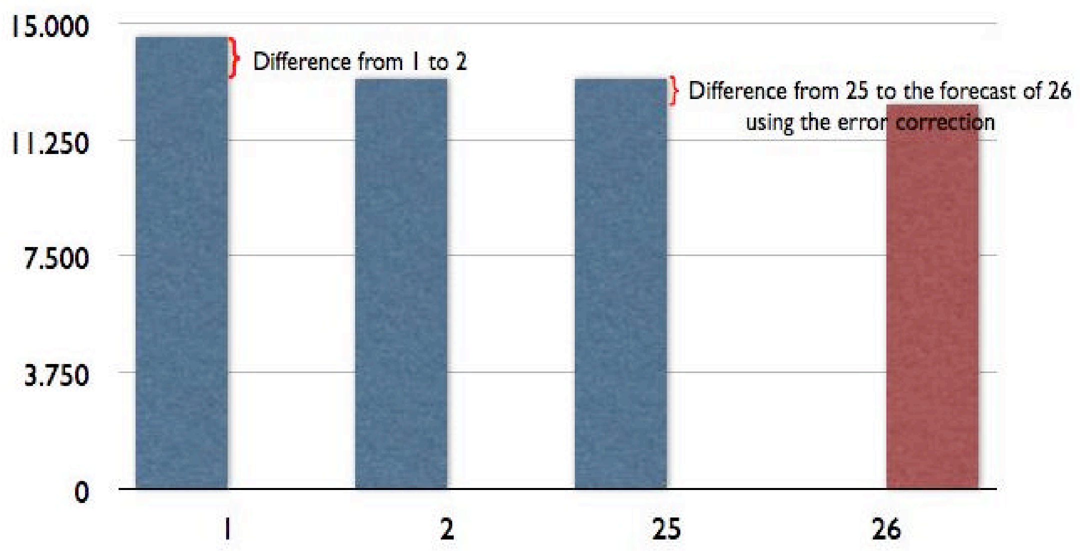

Figure 3.22: Two couples distant L compared.

Because the series is not always stationary we use a parameter in front of λ, this parameter is ρ, and it stays between 0 and 1.
This is the complete expression of the forecasting algorithm:

$$F_{t+m} = (S_t + mb_t)I_{t-L+m} + \rho\lambda_{t+m}$$

The choice of the parameter ρ could be done using the experimental way as for α, β, and γ. This way is the most performance one, but the training dataset have to be similar in shape to the future values.
This is the pseudo code:

```
while $\rho$ <1
while $\alpha$ <1
          while $\beta$ <1
               while $\gamma$ < 1
                    for t =0 until t < length of training dataset
                              calculate the forecasted value
                         if forecasted value is a θ-approximation of the real value
                              true++
                              else
                                   false++
                         if true > stored-value
                              save new parameters
               $\gamma$=$\gamma$+0,01
          $\beta$ = $\beta$+0,01
     $\alpha$ =$\alpha$+0,01
$\rho$=$\rho$+0,01
```

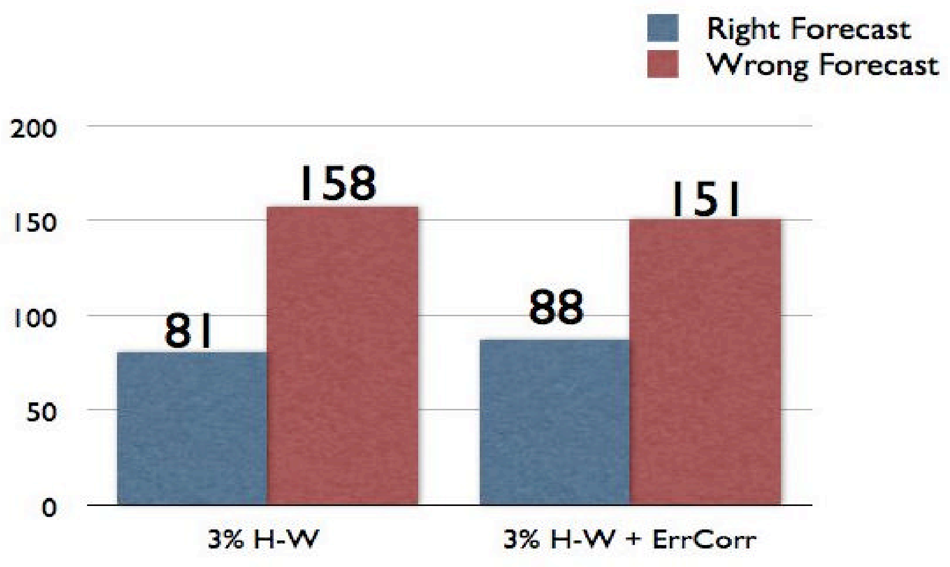

Figure 3.23: H-W vs. H-W with Error Correction, error permitted 3%. Time steps

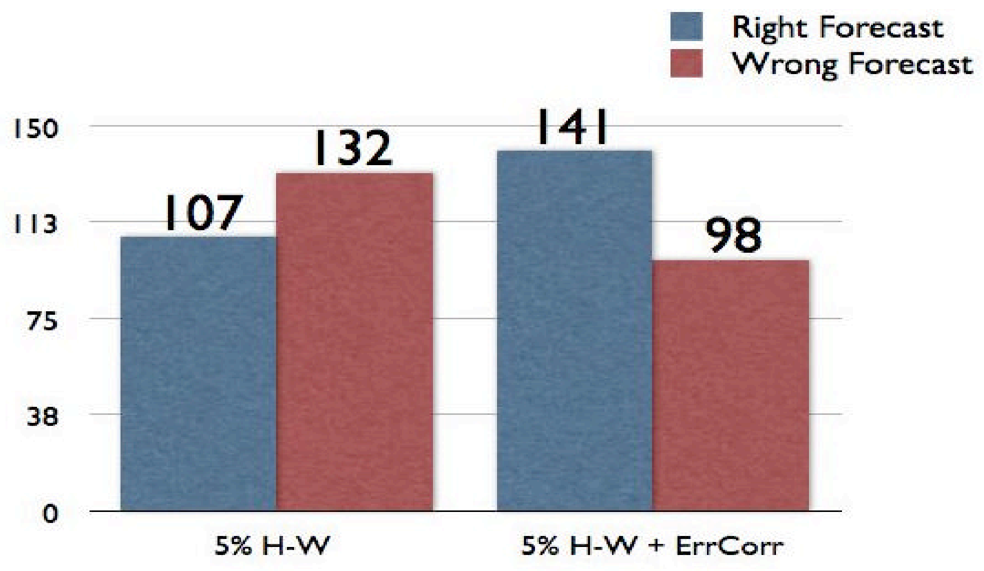

Figure 3.24: H-W vs. H-W with Error Correction, error permitted 5%. Time steps

In the two charts above we can see how the error correction effect the forecasted values in the case of using time steps. The performance increase in relevant percentage for higher error permits, in particular:

- 8.6% using θ=3%, 88 right prediction with the Holt-Winter algorithm with the error correction against 81 using the standard Holt-Winter model

- 31% using θ=5%, 141 right prediction with the Holt-Winter algorithm with the error correction against 101 using the standard Holt-Winter model.

With this final forecast model using the testing data and θ=5% I can predict ,respecting the fixed error bounds, six update every ten updates, and is a good result considering that I using fixed time steps I have no assurance on the stability of the distribution ad happen with a global stable cumulative distributions.

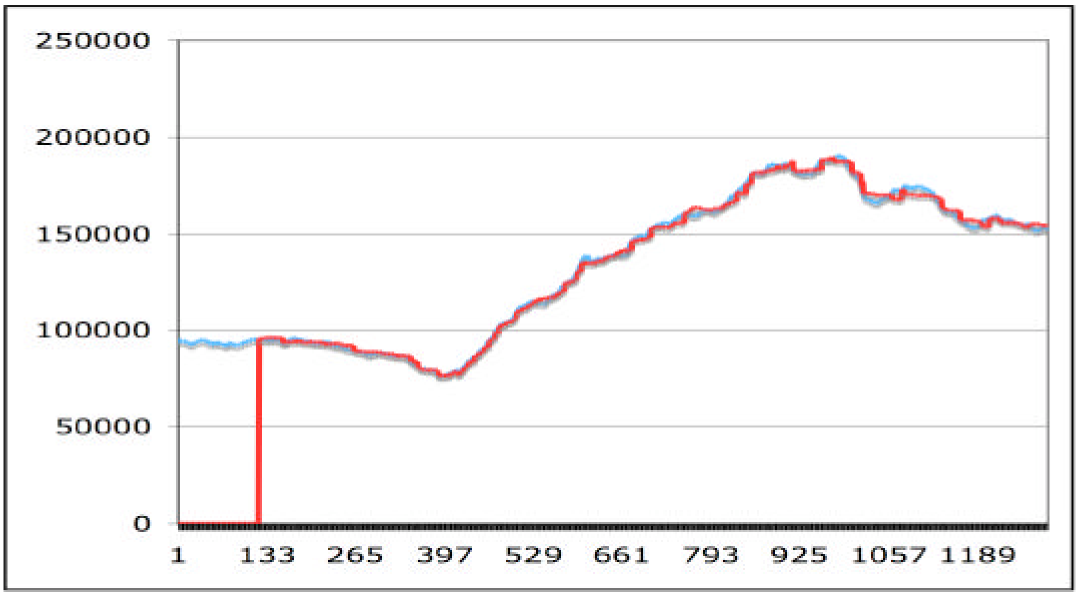

Figure 3.25: results with MWP plus H-W, error permitted of 3%. Sliding window

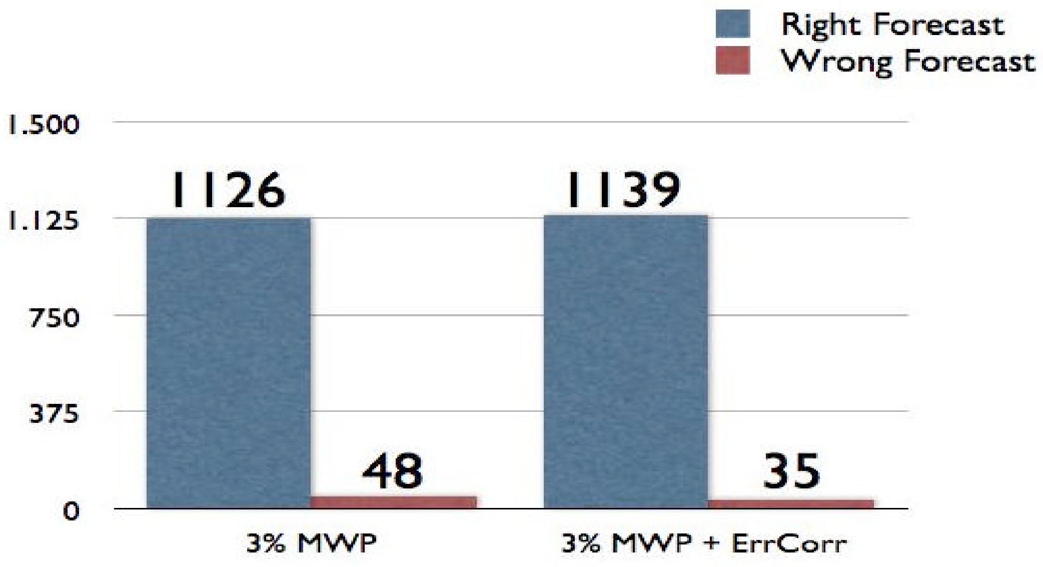

Figure 3.26: H-W vs. H-W with Error Correction, error permitted 3%. Sliding window

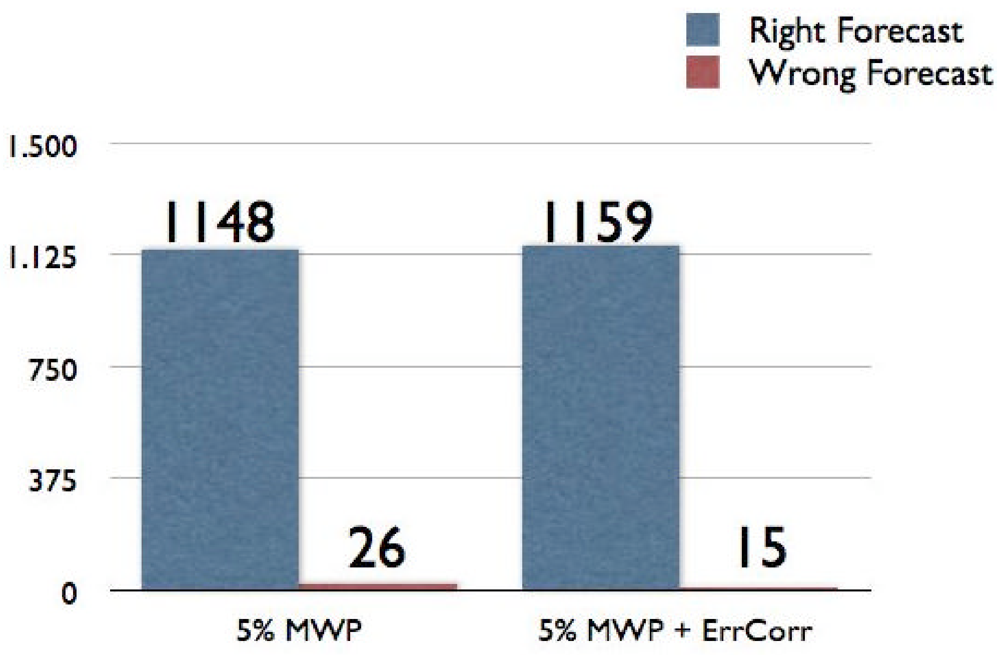

Figure 3.27: H-W vs. H-W with Error Correction, error permitted 5%. Sliding window

The error correction added to the Holt-Winter model. added reduce almost to 0 the number of the wrong prediction using the sliding windows, thanks to the stability of the stream. The performance are:

- 28% less wrong predictions using θ=3%, 54 right prediction with the MWP with the error correction against 48 using the old predictive model
- 43% less wrong predictions using θ=5%, 31 right prediction with the MWP with the error correction against 26 using the old predictive model

3.5 Forecasting Absolute Rank

As in Cormode et al. [**10**], I estimate the absolute rank of v using this formula:

- $\hat{r}(v) = \sum_j \frac{r_j^p(v_{i',j}) + r_j^p(v_{i'+1,j})}{2}$

I have changed the way for assure a θ-approximate N_j at the coordinator site , and I introduce time period, the formula above become:

- $\hat{r}_t(v) = \sum_j \frac{r^p_{j,t}(v_{i',j}) + r^p_{j,t}(v_{i'+1,j})}{2}$

Theorem 3.1 : Assume that, for each remote site j and local quantile value $v_{i,j} \in Q(S_j)$ at time t we have $\left| r^p_{j,t}(v_{i,j}) - r_{j,t}(v_{i,j}) \right| \le \theta N_{j,t}$, and $(r^p_{j,t}(v_{i+1,j}) - r^p_{j,t}(v_{i,j})) \le 2\phi N_{j,t}$, i.e., the range between two consecutive predictions is upper bounded by $2\phi N_{j,t}$. Then for any value of $v \in [U]$, the absolute-rank estimate $\hat{r}_t(v)$ at the coordinator site is an (θ+φ) approximation of the v's true absolute rank $r_t(v)$ in S; that is:

- $r_t(v) - (\theta + \phi)N_t \le \hat{r}_t(v) \le r_t(v) + (\theta + \phi)N_t$

Proof: Note that absolute ranks for *v* across different remote site are clearly additive, i.e. the overall rank is the summation of the per-site ranks, and , by definition of bounding quantile values, we have $(v) \in (r_{j,t}(v_{i',j}), r_{j,t}(v_{i'+1,j}))$, thus, we have:

$$r_t(v) = \sum_j r_{j,t}(v) \ge \sum_j r_{j,t}(v_{i',j})$$

$$\ge \sum_j \left[r^p_{j,t}(v_{i',j}) + (r_{j,t}(v_{i',j}) - r^p_{j,t}(v_{i',j})) \right]$$

$$= \sum_j \frac{r^p_{j,t}(v_{i',j}) + r^p_{j,t}(v_{i'+1,j})}{2} - \sum_j \frac{r^p_{j,t}(v_{i'+1,j}) - r^p_{j,t}(v_{i',j})}{2} + \sum_j (r_{j,t}(v_{i',j}) - r^p_{j,t}(v_{i',j}))$$

Based on the definition of $\hat{r}_t(v)$ estimate and the assumption of the theorem, this last inequality gives:

$$r_t(v) \ge \hat{r}_t(v) - \sum_j \phi N_{j,t} + \sum_j (r^p_{j,t}(v_{i',j}) - r_{j,t}(v_{i',j}))$$

$$\ge \hat{r}_t(v) - \phi N_t - \sum_j \left| r_{j,t}(v_{i',j}) - r^p_{j,t}(v_{i',j}) \right|$$

$$\ge \hat{r}_t(v) - \phi N - \sum_j \theta N_{j,t}$$

$$\ge \hat{r}_t(v) - (\phi + \theta)N_t$$

The other direction is symmetric.

We should note here that, in principle, it is possible to allocate a distinct total-error threshold $\varepsilon_j = \phi_j + \theta_j$ to each remote site *j*=1,…,*k*. However, as can be seen form the proof of the theorem above, the error guarantee ε at the coordinator site is determined by the maximum per site error, i.e., $\varepsilon = \max_j\{\varepsilon_j\}$. Thus, since the per-site communication cost is clearly monotonic decreasing in its error tolerance, my algorithms allocate the same error $\varepsilon = \phi + \theta$ across all remote sites [10].

v	**0**	**43**	**102**	**141**	**163**	**169**	**183**	**208**	**236**
r(v)	650	4425	5703	8701	11327	14123	16591	19826	22609

v	**269**	**338**	**402**	**436**	**520**	**635**	**669**	**766**	**784**
r(v)	25625	28222	31176	33874	36851	39522	42425	45227	48176

v	**828**	**867**	**892**	**915**	**934**	**965**	**994**	**1005**	**1056**
r(v)	51528	53723	56725	59471	63522	64976	68301	71006	73425

v	**1189**	**1251**	**1362**	**1504**	**1647**	**1927**	**2122**	**2264**	**2557**
R(v)	76452	79301	81900	85754	87538	127222	93203	95953	98875

v	**2843**	**2909**	**3553**	**3773**	**4486**	**5319**	**6507**	**8389**	**8862**
r(v)	102076	104422	107377	110177	112903	115702	119603	122379	126126

v	**9118**	**11126**	**12592**	**18278**	**23521**	**2020644**
r(v)	127175	129827	132677	135456	139058	142425

Table 3.2: Absolute rank values of S_t, ϕ=*0,02.* Time steps

The first thing to observe in Table 3.1 is that the absolute rank of the first value is not 0, this for the definition of rank, but the absolute-rank prediction formula set it to zero by calculating it as $r^p_{j,t}(v_{i,j}) = i\phi(N_{j,t})$ with *i=0.* This is a big assumption for the algorithm because if the rank of the lowest value is more than $(\phi + \theta)N_{j,t}$ the rank predicted with the Cormode et al. [10] will be wrong.

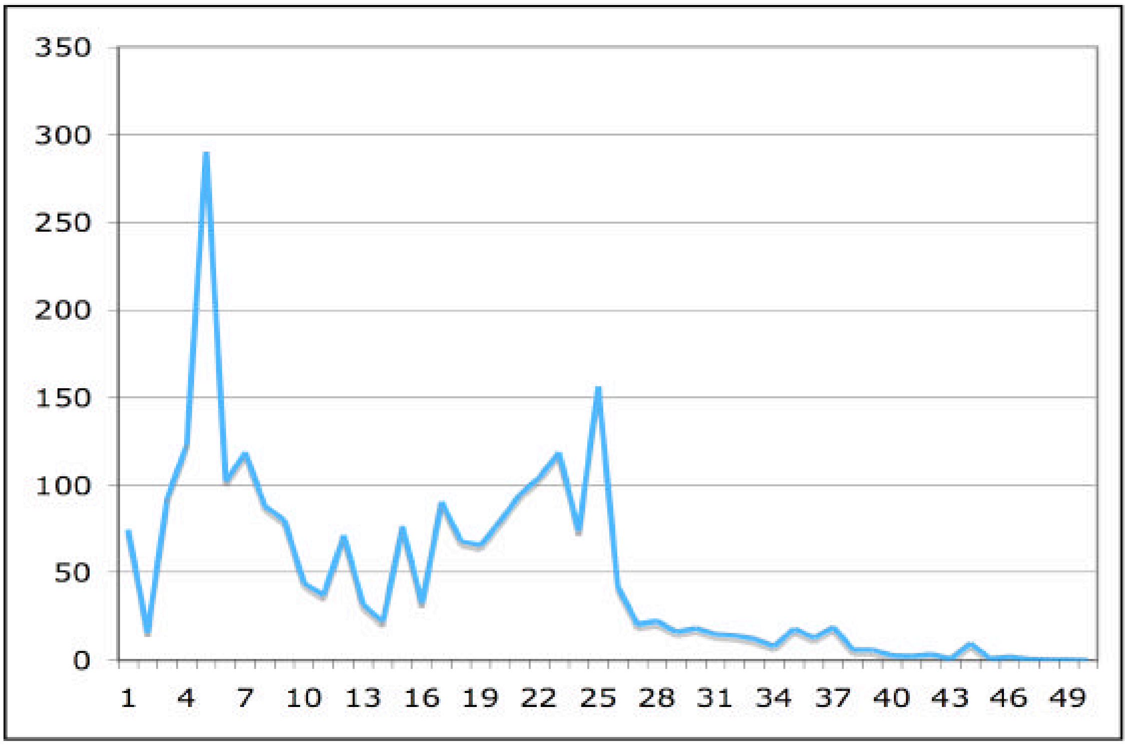

Figure 3.28: Average multiplicity in the interval between two quantile

The Figure above shows the mean number of request for each value in the interval from one quantile and another; in other words, the graphs represent the quantile rank divided by the number of different elements present in the complete distribution in the interval form one quantile and the other. The plot shows the non linearity of the distribution, for example, in the first interval, where the values of the size are low, i.e. static html pages, there are a lot of hits for each value, in the last part of the distribution for each different value in the interval we have a low multiplicity, i.e. big files.

When the distribution of the values inside the global distribution is not linear, as is shown in the figure above, that can happen often at the extremes of the distribution, the formula for the calculation of $\hat{r}_t(v)$ provided by Cormode et al. [10], $\hat{r}_t(v) = \sum_j \frac{r^p_{j,t}(v_{i',j}) + r^p_{j,t}(v_{i'+1,j})}{2}$, is not the best choice because if $v_{i',t}$ and $v_{i'+1,t}$ are very far or very near from each other respect the normal distance between two values in the rest of the distribution, the absolute-rank value predicted at time t of a value v, near to $v_{i',t}$ or to $v_{i'+1,t}$ may will be too far form the real value. How we can see form the figure above the average multiplicity of the values between two quantile values change a lot inside the *Q(S)* so is necessary an adaptive formula for the calculation of the absolute rank prediction of value *v*.

I develop a new formula for calculate the absolute-rank prediction at the coordinator site:

$$\hat{r}_t(v) = \sum_j \frac{r^p_{j,t}(v_{i'+1,j}) - r^p_{j,t}(v_{i',j})}{v_{i'+1,j} - v_{i',j}}(v - v_{i',j}) + r^p_{j,t}(v_{i',j})$$

This formula makes the predicted absolute-rank of v more precise because is proportionally near to $r^p_{j,t}(v_{i',j})$ or $r^p_{j,t}(v_{i'+1,j})$ if *v* is nearer to $v_{i',j}$ or $v_{i'+1,j}$.

Another problem of the Cormode et al. [10] algorithm is that if the global distribution contains a value with a multiplicity more than ϕN or ϕN_j, for the site case, the second condition of the theorem 3.1 may not be satisfied, this situation could happen if ϕ is too small and the distribution does not contain a lot of different values, or if we have heavy-hitter values. This because the absolute rank is defined as $r(v) = |\{u \in S : u \leq v\}|$ so if the multiplicity of *v* is more than ϕN or ϕN_j we have an high probability of two consecutive *v* in the *Q(S)* or $Q(S_j)$ with the same rank and the predicted absolute-rank of this two values calculate at the coordinator site will be very form each other because the Cormode et al. basic formula for the predicted absolute rank is $r^p_j(v_{i,j}) = i\phi(N_j + \delta_j t_j)$. My N_j or $N_{j,t}$ is already a predicted values. In this case will be:

- $r^p_{j,t}(v_{i,j}) = i\phi(N_{j,t})$ for the nodes side.
- $r^p_t(v_i) = i\phi(N_t)$ for the coordinator side.

The only way to correct this simple formula is to use something else instead of the $i\phi$ element.

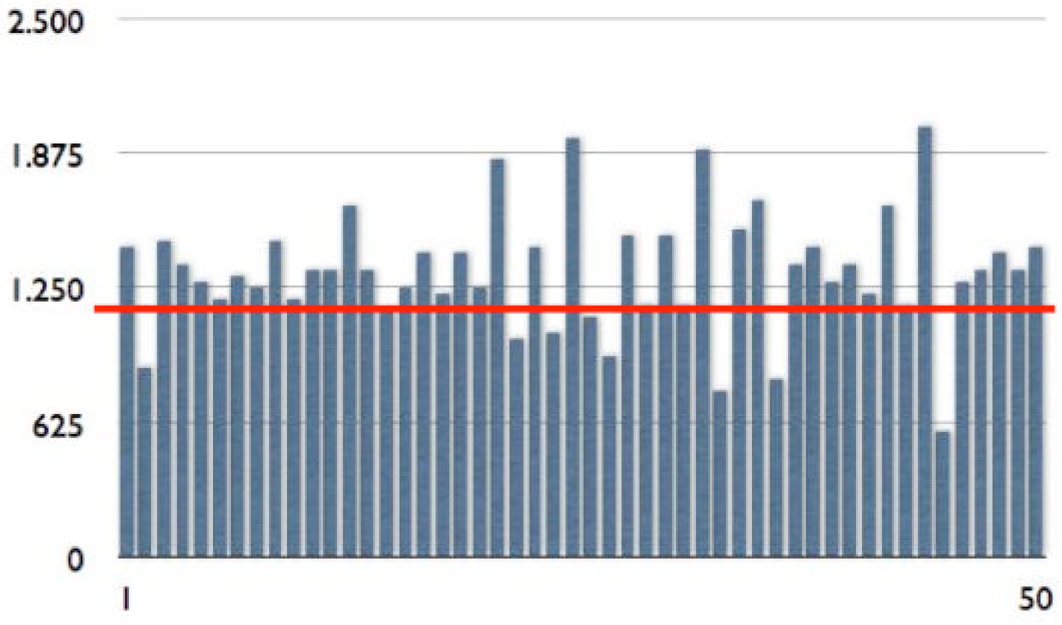

Figure 3.29: result of $(r(v_{i+1}) - r(v_i))$ *in Q(S)*

Ideally each difference of successive values' absolute-rank may result ϕN_t in the graph above the value pf ϕN_t is marked by the red line. Is easy to see than same value double the ideal value and others stay a lot behind. For resolve this problem the way is to find another formula that is not static, and that could follow the distribution evolving.

In particular for using the Holt-Winter model the first two L series of update are used for calculate the initial values of the model, so the updates of $N_{j,t}$ and $Q(S_{j,t})$ are transmitted form the nodes to the coordinator, the core idea is to make i χ values for each node that represent the historical average of the absolute-rank percentage of the i values inside $Q(S_{j,t})$.

These values are:

- $\chi_{i,j} = \frac{\sum_{t=1}^{2*L} \chi_{i,t,j}}{2*L}$ for the nodes side.
- $\chi_i = \frac{\chi_{1,i} + \chi_{2,i} + \ldots + \chi_{k,i}}{k}$, where k is the number of the nodes in the SN, for the coordinator side.

So the new formulas for the prediction of the absolute rank are:

- $r^p_{j,t}(v_i) = \frac{\chi_{i,j}(N_{j,t})}{100}$ for the nodes side.
- $r^p_t(v_i) = \frac{\chi_i(N_t)}{100}$ for the coordinator side.

A absolute-rank prediction could be wrong for:

- bad forecast of $N_{j,t}$
- bad choice of $\chi_{i,j}$

These two event could happen together or not, and for saving in the communication from the nodes and the coordinator know what is wrong is essential. The cost for update the forecast model of $N_{j,t}$ is lower than the cost f updating the entire χ_j array . So at the node side first there is a check of $\left|r^p_{j,t}(v_{i,j}) - r_{j,t}(v_{i,j})\right| > \theta(N_{j,t})$ if this condition is true, the node try to recalculate $r^p_{j,t}(v_{i',j})$ with the use of $N_{j,t}$ instead of $\hat{N}_{j,t}$, then recheck the condition above, if it is true compute a quantile summary Q(S$_j$) composed by the v$_{i,j}$ quantile values and the respective $\chi_{i,j}$ values and send it together with $N_{j,t}$ to the coordinator; if the condition is false only $N_{j,t}$ is sent to the coordinator.

Pseudo code for this procedure:

```
GoodpredictionsN=true
GoodpredictionsQ=true
For i=0 to $\frac{1}{\phi}$ do
        If $\left|r^p_{j,t}(v_{i,j}) - r_{j,t}(v_{i,j})\right| > \theta(N_{j,t})$ then
            GoodpredictionsN=false

If GoodpredictionsN == false
    Recalculate new $r^p_{j,t}(v_i)$ using $N_{j,t}$
        For i=0 to $\frac{1}{\phi}$ do
            If $\left|r^p_{j,t}(v_{i,j}) - r_{j,t}(v_{i,j})\right| > \theta(N_{j,t})$ then
                GoodpredictionsQ=false
```

```
If GoodpredictionsQ == false
        Compute new local $\phi$-quantile value summary $Q(S_j)$ and new $\chi_{i,j}$
If GoodpredictionsQ == false && GoodpredictionsN == false
        Send to Coordinator: {$Q(S_j)$, $\chi_{i,j}$, $N_{j,t}$}
        Else if GoodpredictionsN == false
                Send to Coordinator: {$N_{j,t}$}
```

For the testing of this algorithm I use the simulator described in *Chapter 4*. I used the World Cup '98 [**26**] dataset, distributed with a random function over four nodes.

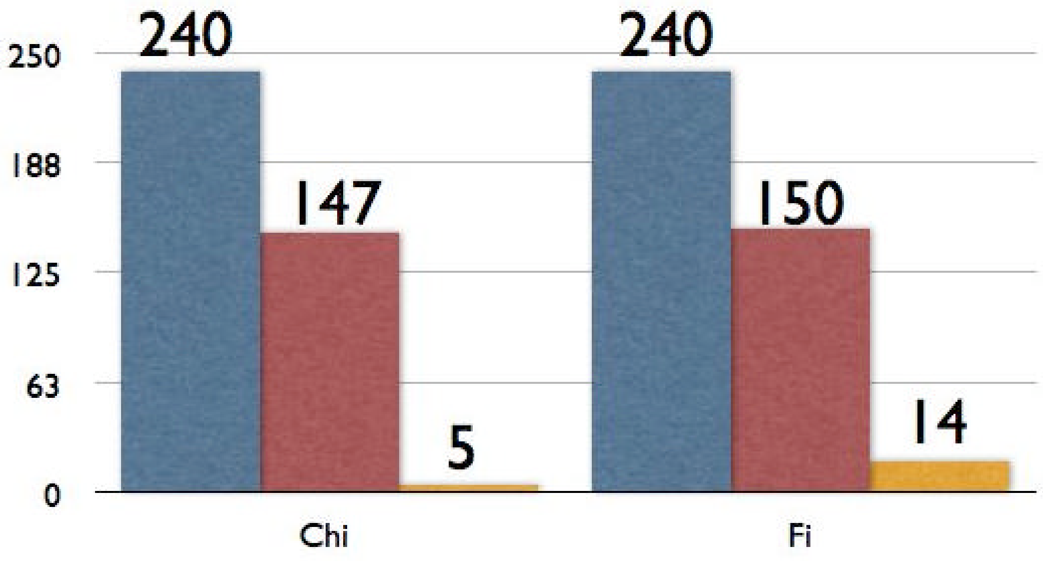

Figure 3.30: Number of communications,ϕ*=0.02,* θ*=0.03. Time steps*

The figure above represent the number of communication per type with two different formula for $r_t^p(v_i)$, the blue column represent the total number of the time steps evaluated, 240 for ten days, not twelve because the first two days, two *L* period are used for the initial values of the Holt-Winter model, the red column represent the total number of communications that occurred from the node to the coordinator, and the yellow column is the number of communications that are complete, so with *Q(S_j),* $\chi_{i,j}$and $N_{j,t}$. These values are the average of the four measured by the nodes.

The labels stand for the algorithm used for the computation of the predicted-rank values:

- Chi: $r_t^p(v_i) = \dfrac{\chi_i(N_t)}{100}$
- Fi: $r_t^p(v_i) = i\phi(N_{j,t})$

The result of this experiment is evident, the research of what parameters make wrong the prediction of the absolute-rank reduce the cost of the communication because the node will transmit to the coordinator only the necessary information to compute ε-approximate ranks.

Also the adoption of a new formula for compute the for the compute the absolute-rank prediction helps to keep low the communication cost making more precise the forecast of the ranks' value.

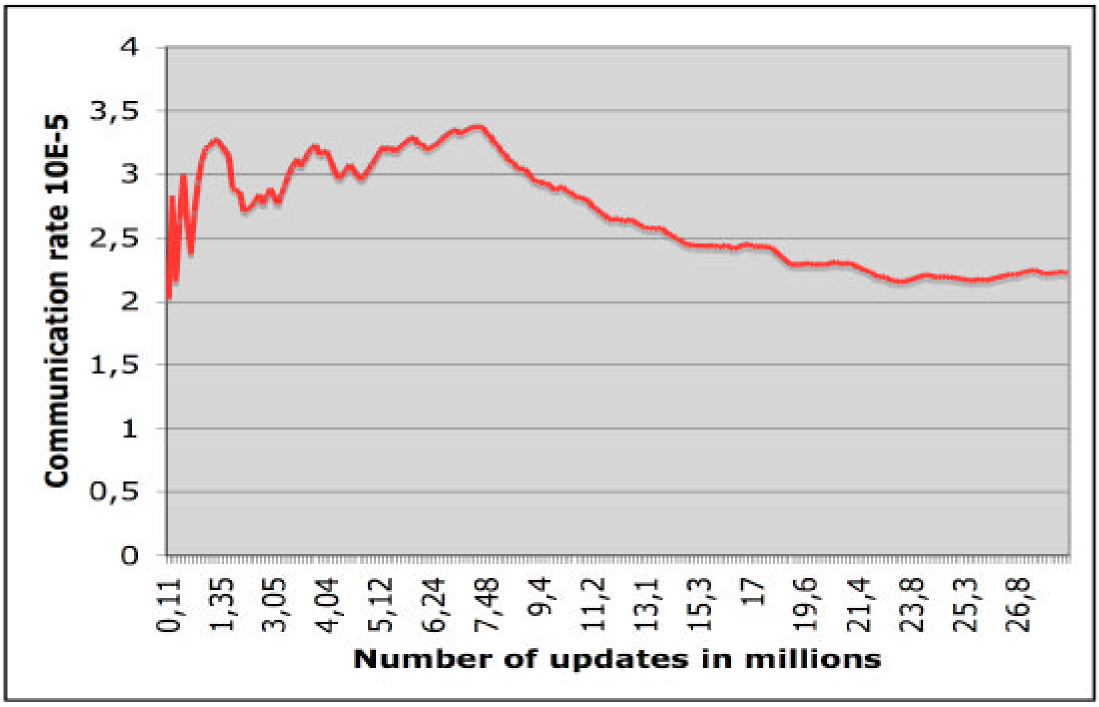

Figure 3.31: Communication rate, ϕ=0.02, θ=0.03. Time steps

The Figure above shows the communication rate for number of updates in the whole system with four nodes and shows the communication rate at the beginning is very unstable and then tend to a constant value. At the beginning we find a peak of one communication from one node to the coordinator every thirty thousand of updates, at the end this value is stable around one update every forty-five thousand updates. This is for the low margin of error permits. I this case we have the guarantee of a ε=5% for the quantile rank

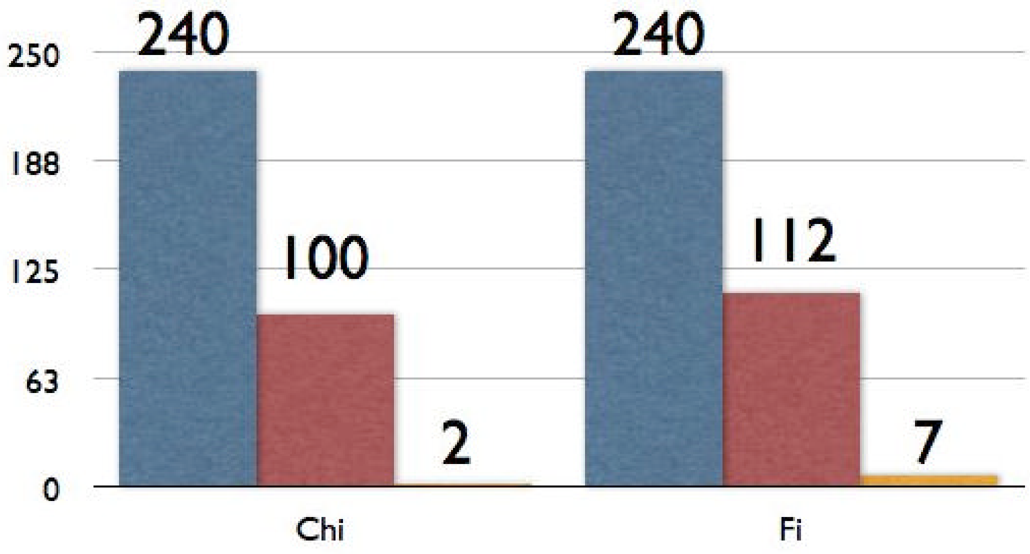

Figure 3.32: Number of communications,ϕ=0.02, θ=0.05. Time steps

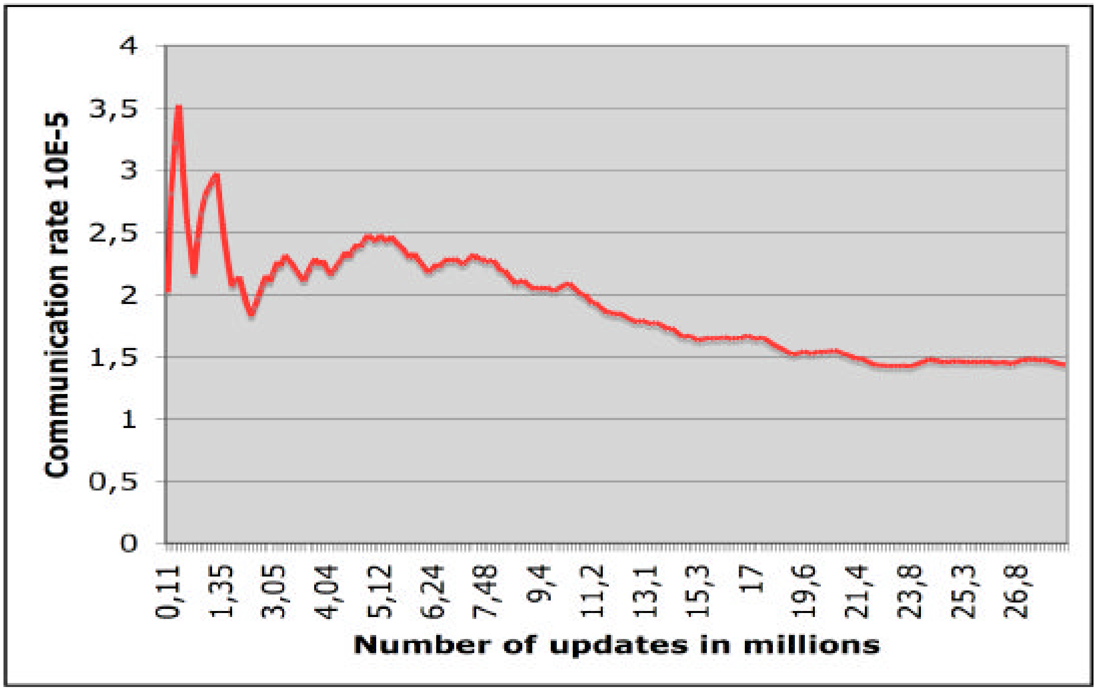

Figure 3.33: Communication rate, ϕ=0.02, θ=0.05. Time steps

The high margin on θ permits to obtain very few communication involving the quantile summary $S(Q_{j,t})$. Reduce this type of communication is important as much as reduce the total number of communications because the cost of communications of the quantile summary depends from the value of ϕ and normally is much bigger than the cost of transmitting only $N_{j,t}$.

Looking at Figure 3.33 we can see that the shape of the communication rate curve with θ=0,05 is the same of the curve with θ=0.03, the only thing that changes is the final stable values of one communication every seventy thousands updates. The high number of updates for communication is the consequence of an higher value of θ.

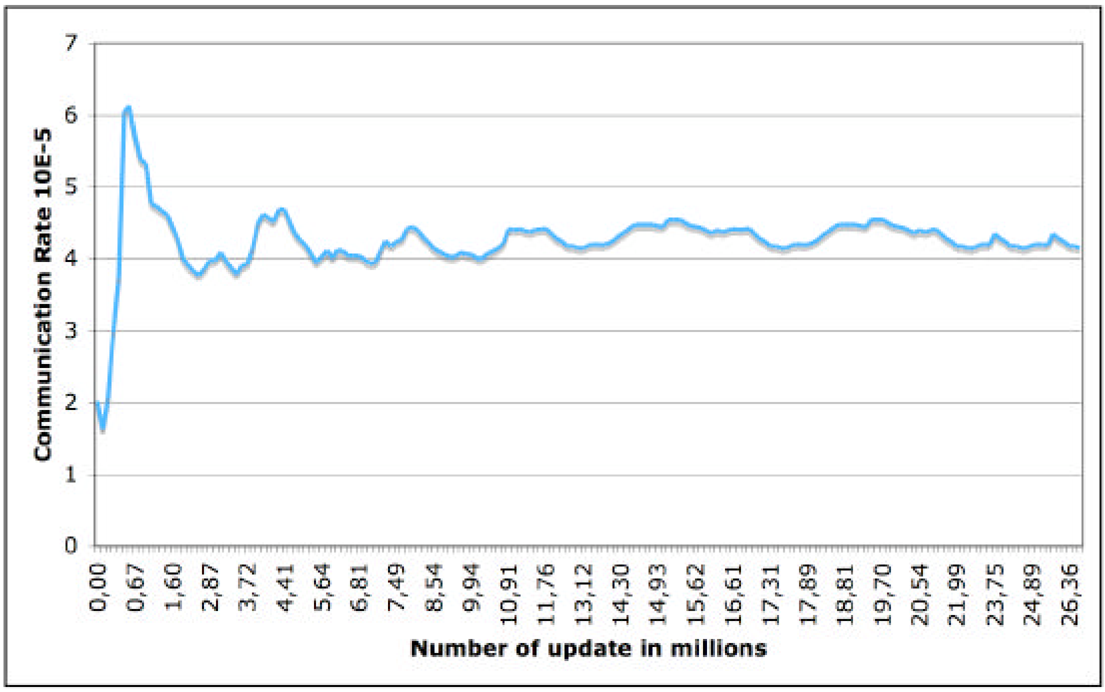

Figure 3.34: Communication rate, ϕ=0.02, θ=0.05. Sliding window

The figure above shows the communication rates for the number of update receives by the node using the sliding window instead of the time steps. The rate is higher using the sliding windows because there is a check every time the window slide, at the beginning there is a communication every fifteen thousand values, at the communication rate is stable around one communication every thirty thousands updates.

One of the main problems in my analysis was the selection of the right time steps for my data. I found two different possibilities:

- Fixed time steps, i.e. one hour, one minute, two hours…
- Sliding window that contains several time steps, i.e., window contains sixty time steps long one minute each.

The main difference from these two possibilities is the precision of the absolute-rank's forecast in the current time steps. If we chose a time step of one hour, we have (θ+ϕ)approximate continuous-monitor queries answers for the past time steps, but not for the current because the nodes may update and correct the coordinator behavior only at the end of the time step. In most of cases the current behavior could be correct but in some case may not. My test show that if the coordinator behavior is not correct the error is bounded of 8%-10% of N_t.

If we chose to use the sliding window, we have *(θ+ϕ)*-approximate continuous-monitor queries answers also for the current period, but the communications costs of a system that use sliding windows with a length of *n* time steps are higher than the communication costs of the same system using a time step long *n* time steps.

Because with the sliding windows the performance of my prediction algorithm for forecasting of $N_{j,t}$ are higher thanks to the stability property of the sliding window, but not *n* times the performances of the algorithm with the time steps. So if we look to save at almost any cost the communication costs the time step is the better choice, if we look at the guarantee of an (θ+ϕ)-approximate answer at the continuous-monitor queries the sliding window is better.

3.7 Threshold

There is a class of fundamental problems called "thresholded counts", in the SN and in the WSN, where we must return the aggregate frequency count of an event that is continuously monitored but distributed nodes with a user-specified accuracy whenever the actual count exceeds a given thresholds value. Consider that queries form different domains:

- Rise an alert when the total number of people in an hall exceeds 400 and report the number of people detected
- Rise an alert if the number of connection to a server inside the network are more than 20000

(from [**31**]) Which users within the monitored network receive more than thousand different connections?

In each case there are two parts to the query: a request for a sum or count of particular quantity, and a minimum threshold. In almost every application involving measuring quantiles like these, it is only important to know the quantiles when they exceed a specified level. In general the thresholds can be specified either as a port of the query, or learned by the system in response to the observed data. The second component of these types of queries is to return a count of a particular set of values, but normally an exact result is not needed, is the value is higher than the threshold, a strong approximation of how much is higher is in most of cases sufficient. For example if a user receive more than thousand connection, to know exactly f the connections are 1234 or 1324 is not important, is important to know the number of connections with a certain error guaranteed bound, we call this bound ε. In continuous monitoring queries we are interested in decide if N_t at the coordinator site is grater than a threshold τ, the exact may be difficult and always involve much communications, because for every change in the $N_{j,t}$ at node site the node have to send an update to the coordinator. The best way for reduce the communication cost and make possible this threshold monitoring is too use approximate answer, in particular ε-approximate answer. If N_t is greater than $(1-\varepsilon)\tau$, the coordinator should report an alarm.

The goal is to reduce the communication cost also in this kind of threshold continuous monitor queries, using the algorithms that I developed for the quantile tracking problem. The first hypothesis is to use two different algorithm for the execution of this kind of queries. The first algorithm that I want to use is developed by Yi [32] that ensure low communication cost between the nodes and the coordinator until $N_t \leq (1-\varepsilon)\tau$. If $N_t > (1-\varepsilon)\tau$ I use the Holt-Winter model for compute a strong ε-approximation of N_t.

The Yi model [32] is based on an additive stream but can be easily expanded to use sliding window. The environment is composed by k nodes and one coordinator, the threshold is set to τ, the error permitted to ε, and the number of round to h equals to one at the beginning. The coordinator set a temp threshold $\sigma = \frac{\tau}{h*2*k}$, and send it to the nodes. A node sent a signal to the coordinator every time it receives σ elements. The coordinator terminate the round after it receives k signal form the nodes and then add 1 to h and recalculate σ. At this point the coordinator knows that N_j is more than $\frac{\tau}{2}$ but less than τ, this value of N_j at the coordinator site is good because our interest is to track N_j only if it is grater than τ. Now each node sends a signal every $\sigma = \frac{\tau}{4k}$ elements. The process continues until the round in which each site sends out a signal every $\frac{\varepsilon\tau}{2k}$ elements, and at the end of this last round we can say that N_j is in $\left[(1-\varepsilon)\tau,\tau\right]$.

This algorithm only works with insertion of elements, so no deletions are allowed so is impossible to use a sliding window. With small modifications this algorithm can manage also deletions of elements. This is the pseudo code of the algorithm that allows also deletions.

Node:

```
Current= is the value for the coordinator of the sliding window at the node side
For each update of the sliding window
Value= sliding_window
If(OutOfBound==false)
If(value>current+tempthresold)
            Current = current + tempthresold
            SignalUP()
             //Send to coordinator a signal that the window receives    temp
            threshold   updates

If(value<current)
            Current = current - tempthresold
                   SignalDOWN()
```

```
                    //Send to coordinator a signal that the window is smaller than
                    the current coordinator's behavior
```

Coordinator:

```
Updates=number of signals send by node for each round
K= number of nodes
Round=1 //number of rounds
Global= current value for the coordinator of $N_j$
SignalUP()
{
    updates++
    Global= Global+ tempthresold
    If(Global== $(1-\varepsilon)\tau$)
        Alarm_rise()
    if(updates==K)
        round++
        tempthresold = $\sigma = \frac{\tau}{round*2*k}$
        updates=0
}

SignalDOWN()
{
    updates--
    Global= Global - tempthresold
    if(updates== - K)
        round--
        tempthresold = $\sigma = \frac{\tau}{round*2*k}$
        updates=0
}
```

There are two kind of signal, SignalUP and SignalDOWN for managing the insertion an the deletion. So the first part of the answer for general threshold query is done. The goal now is to track the values of N_t when it exceeds the threshold τ.

When the coordinator rise the alarm also have to send to the nodes a request of the necessary parameter for predict ε-approximate $\hat{N}_t$ values using the Holt-Winter algorithm with the correction of error. So we can track the evolution of the size of the stream with a ε-approximation guaranteed from the algorithm. When $\hat{N}_t$ return under $(1-\varepsilon)\tau$ the coordinator send a message to the nodes with the new σ, and stop the tracking with the Holt-Winter algorithm.

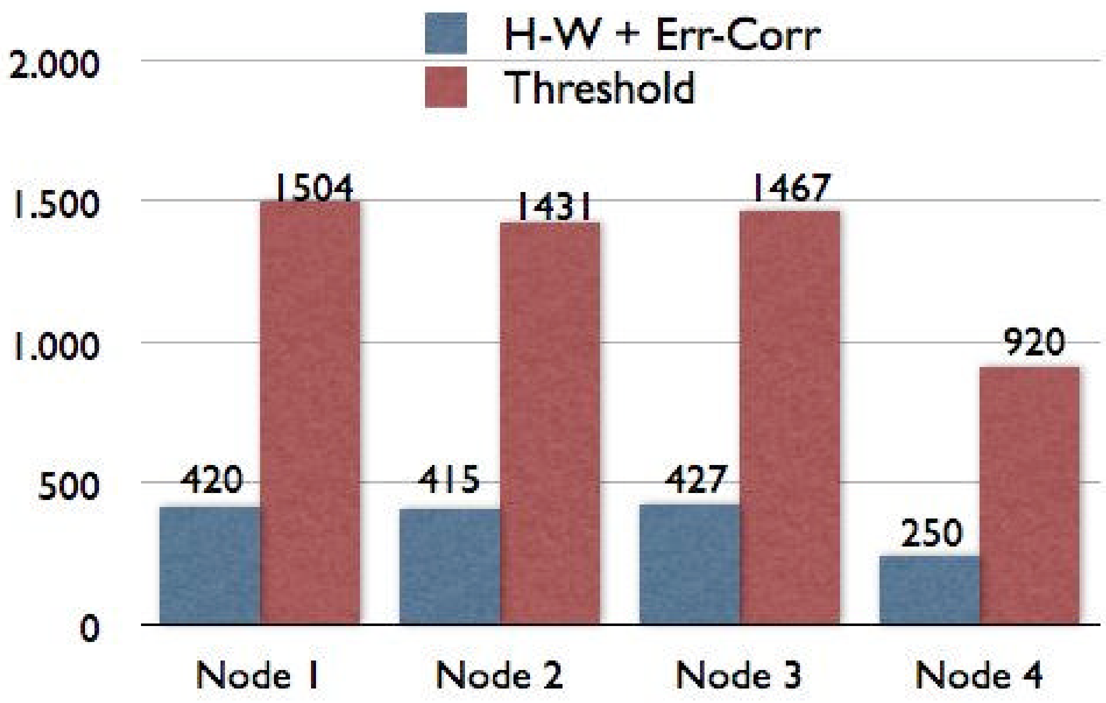

Figure 3.35: Number of communication, threshold: 175000

I run some test with the simulator described in *Chapter 4*, using 4 nodes, the World cup'98 dataset [26] and a random distribution of the elements over the 4 nodes. I obviously used the sliding window method with a length of sixty minute, and a dataset long ten days as in the precedent tests, so more than 28 millions of connection received by the nodes.

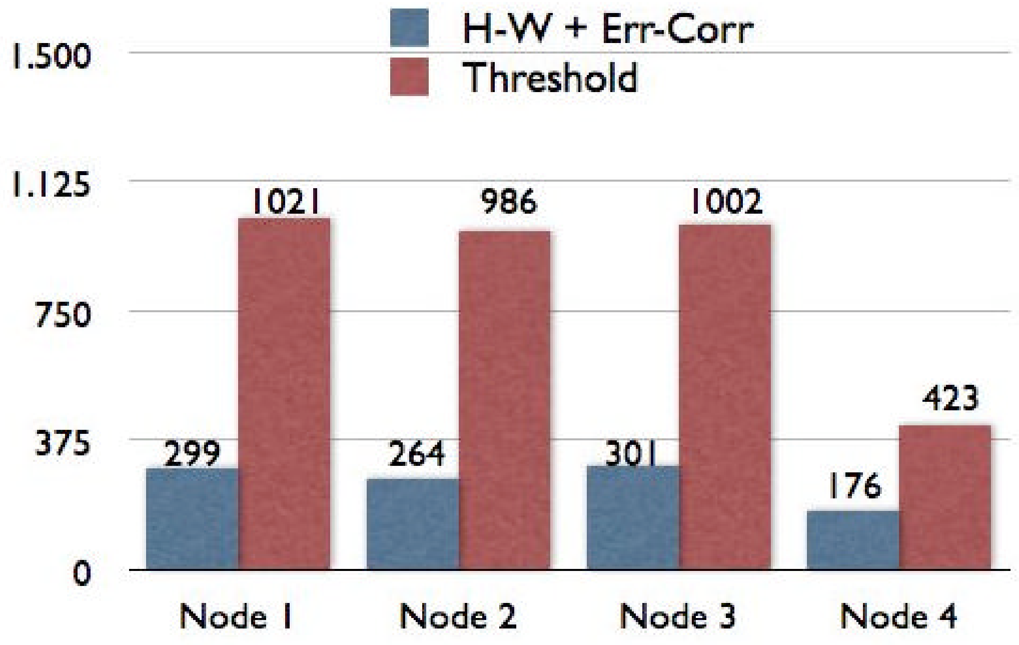

Figure 3.36: Number of communication, threshold: 250000

The two figures above show the number of connection occurred during the test from each site to the coordinator. I use two different threshold values for underline the big difference occurred when I change the threshold. Bigger is the threshold bigger are σ values and so less communications are allowed. An high threshold means also that the number of time that N_t goes over the threshold decrease, and so also the communications made by the Holt-Winter algorithm for the monitoring of N_t decrease. The number of communications generated by the threshold tracking algorithm is strictly correlated to the shape of the curve and the value of the threshold; if the distribution at local goes up and down from a temp-threshold line a lot of communication is made by that site and may a lot of switches to the Holt-Winter model are made too. This changes are very expensive in terms of size of the communication because every time the node have to send to the coordinator the initial values and parameters for the Hot-Winter model.

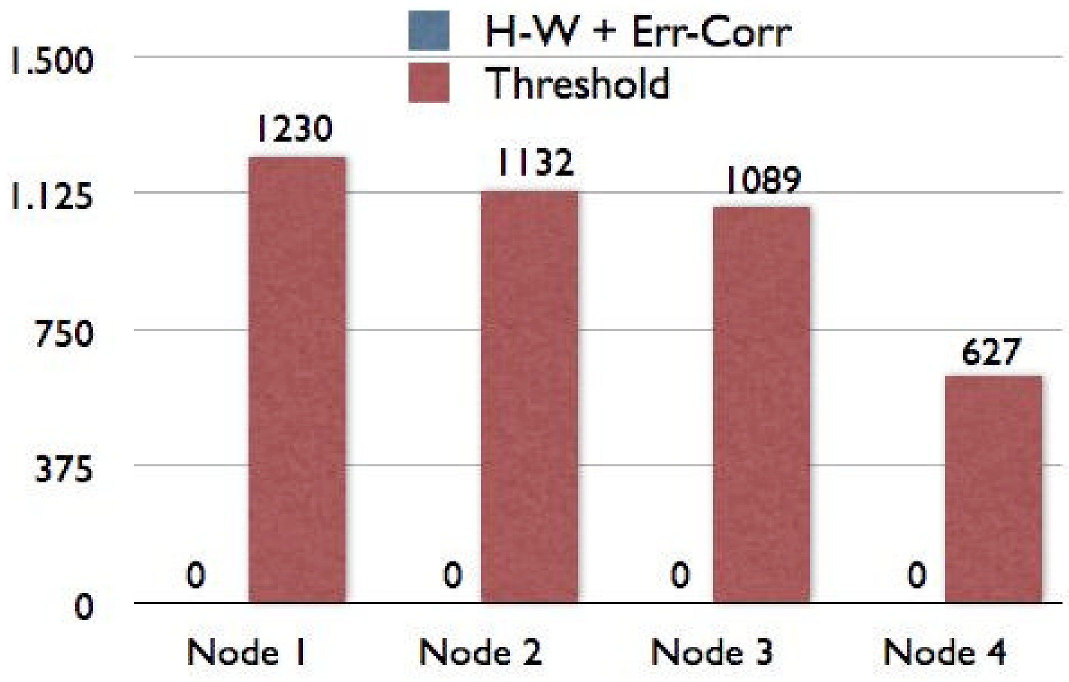

Figure 3.36:Number of communication, threshold higher than the maximum value of N_t

To make the test represented in the Figure 3.36 I set the threshold a bit above the maximum level of N_t observed in the previous tests. The interesting thing is that the communication are a lower than those of the test run with the threshold equal to 300000 but the algorithm runs without interruption by the Holt-Winter one. This happen because the σ values are very high respect that of the test of Figure 3.34.

With figure 3.37 I want to show that in this case if we run the Holt-Winter plus error correction model, on this datasets the number of communications made by this algorithm that could assure strong ε-approximate values of N_t is lower than using the threshold tracking, and also assure best approximation of N_t. For sure is that the cost may be non really lower, because the update to the coordinator of the Holt-Winter model is made by a number , a double in most of the cases, and some times longer update are needed for update parameters of the model. The Threshold tracking algorithm using only two signals from the node to the coordinator. But I think that we have to put this update inside a protocol of communication so maybe a double or two bits can't make the difference in the real size of the composed mainly by protocol necessary information.

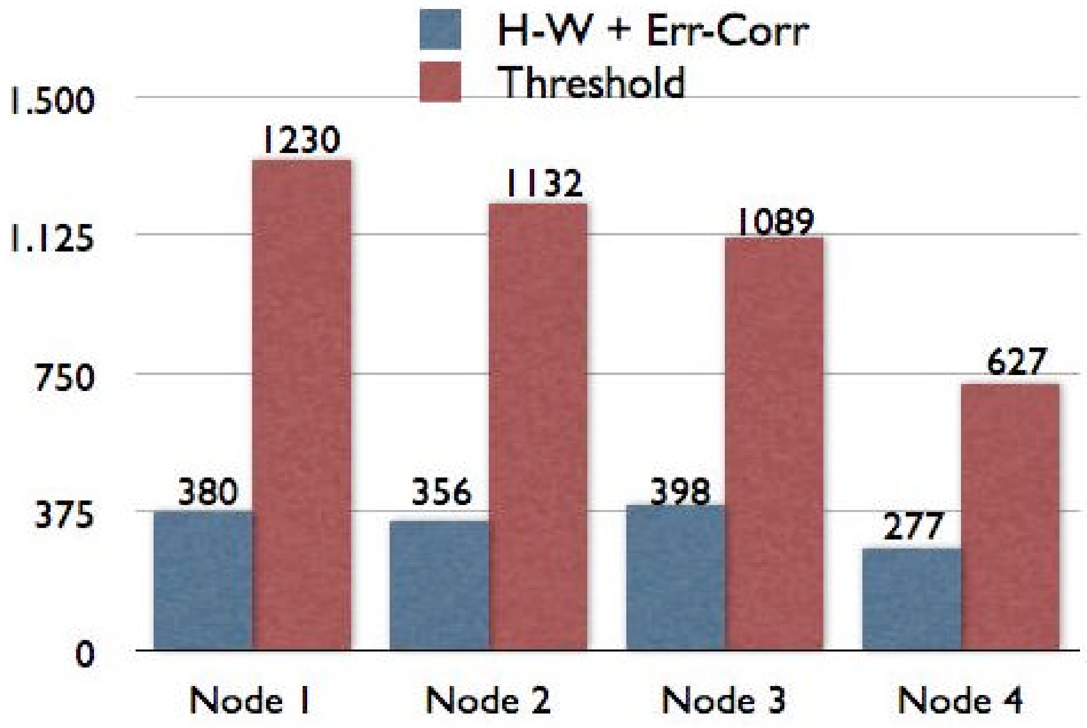

Figure 3.38: H-W vs. Threshold tracking.

With the figure 3.38 I want to show The difference in the number of communication using only the threshold tracking, so using an high threshold for doing the complete tracking without interruption, and using the time-series tracking with the modified Holt-Winter model. This graphs only shows the number of the communication not their size, the size of un update using the Holt-Winter algorithm it's huge if we confront it with the size of the update with the threshold tracking, so in this case less communication may not mean optimization.

4. Simulator Description

4.1 SRS

The SRS document itself states in precise and explicit language those functions and capabilities a software system (i.e., a software application, an eCommerce Web site, and so on) must provide, as well as states any required constraints by which the system must abide. The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it.

It's important to note that an SRS contains functional and nonfunctional requirements only; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

4.1.1 Introduction

- *Purpose*: the purpose of this SRS is to illustrate the function available and the requirements that the simulator of SN ca satisfy

- *Description*: this software is a simulator for sensor networks, it has to keep data from a source as a text file, simulate the environment, and communicate data the nodes. Each node make some calculation and may can communicate with the coordinator.
- Summary: the rest of the SRS is organized as follows:
  - General description
    - Functions
    - Users
    - Restrictions
  - Specifications:
  - Environment
  - Node
  - Coordinator
  - Forecast Engine
  - Frequency Vector

4.1.2 General Description

- Functions: the simulator give this functions, there are explored in section 4.1.3
  - Environment:
    - Collect data from a data-source
    - Send data to the nodes
  - Node:
    - Receive data from the environment
    - Send values to the Forecast Engine for the computation of the initial values of the Forecast model
    - Send parameters to the Forecast Engine for the forecast of the next value.
    - Construct the absolute and quantile rank array
    - Store the values received form the environment in a Frequency Vector

- Retrieve values from Frequency Vector
- Send updates to the coordinator
- Manage the local threshold sending to the coordinator up o down signal
- Interact with the coordinator on threshold managing
- Have the possibility to work with only threshold, only Holt-Winter or a mix of the two.

Coordinator:

- Receive from nodes the updates of the stream, the *S(Q)*, the parameters for the correct forecasting, and the signals for the threshold managing.
- Send to nodes the values for new threshold.
- Interact with the Forecast Engine for get a correct behavior of the values at nodes site
- Manage the threshold problem, and when necessary send to nodes the signals for switching from an algorithm to another
- Have the possibility to work with only threshold, only Holt-Winter or a mix of the two.

Frequency Vector:

- Search and insert
- Sort
- Clear
- Sum or subtract two Frequency Vectors
- Receive values form node and coordinator
- Transform the Frequency Vector in an array
- Count the elements inside

Forecast Engine:

- Use different algorithms for forecast values
- Calculate all the values necessary to the forecasting
- Communicate with nodes and coordinator

- *Users*: the software is used by two “kind” of users: nodes and coordinator:
    - node: A node is a sensor or a wireless sensor that measure some characteristics form the data captured in the environment
    - coordinator: monitor the entire network, interact with the nodes when is strictly necessary in order to safe communication cost

- *Restriction*: This software is only for testing use. Can’t acquire data from the real world, and the only purpose is to simulate the communication between nodes and coordinator using different algorithms in order to find the best solution and implement it in the real world.

4.1.3 Specifications

In this section I analyze in a more detailed way all the functions described in the paragraph 4.1.2 above. The functions are grouped by class.

- Environment:
    - Collect data from a data-source: read a file with a dataset and transform it in data usable by the system.
        - Input: file in normal text format
        - Processing: phrase the data for obtain only the necessary information and pack this information in a structure
        - Output: a structure composed by the time and value of the data.
    - Send data to the nodes: take data from the collect function and send it to a specific node
        - Input: the structure with time and value of the data
        - Processing: a random algorithm choose the node , simulating a real environment, and the structure is sent to the node
        - Output: and ack from the node

- Node:

- Receive data form the environment: Receive data and store it in a Frequency Vector.
    - Input: data structure.
    - Processing: insert the value in the correspondent Frequency Vector.
    - Output: confirmation of insertion in the Frequency Vector.
- Send values to the Forecasting Engine for the computation of the initial values of the forecast model.
    - Input: array of the observed data of length two L.
    - Processing: Computation of initial Trend Seasonality and Smooth arrays.
    - Output: the three initial arrays.
- Send parameters to the Forecasting Engine for forecast next value.
    - Input: the four parameters of the model used, the three array of Smooth, Trend and seasonality and the current time.
    - Processing: Computation of the value at time +1, and calculation of all the values of trend, smoothing and seasonality.
    - Output: the value forecasted for time+1.
- Construct the absolute and quantile rank array: this function is necessary for complete all the task that involve the quantiles.
    - Input: Frequency Vector associated to the node, ϕ, and the kind of rank
    - Processing: Expand completely the Frequency Vector to an array, then search for the values determinate by ϕ and then calculate the rank of that values.
    - Output: an matrix composed in this manner: the first row by the values determined by ϕ, and the second row by the rank relative to the value.
- Store the values received form the Environment in a Frequency Vector: this function keep updated the Frequency Vector associated with the node .
    - Input: Frequency Vector associated to the node, and the value to insert.
    - Processing: search inside the Frequency Vector if the values to insert is already present, if it is, the multiplicity of the element is increased by 1, if it is not present a new insertion is necessary.
    - Output: the Frequency Vector updated.

- Retrieve a value from the Frequency Vector: Given an index of a value the function returns the value and the multiplicity of the value.
  - Input: Frequency Vector associated to the node, and the index.
  - Processing: search inside the Frequency Vector for the value at the input index position.
  - Output: the value and his multiplicity.
- Send updates to the coordinator: every node have the real observation and can compute the forecast value for that observation, that is exactly what the coordinator does, if that prediction does not stay inside the prescript bounds an update to the coordinator is necessary.
  - Input: the real value, and eventually the new parameters for the model.
  - Processing: send to the coordinator the values and eventually the new parameters.
  - Output: ack from the coordinator.
- Manage the local threshold sending to the coordinator up or down signal: for the threshold continuous-monitor queries is necessary that every nodes keep monitoring its local distribution respect the temp-threshold delivered by the coordinator.
  - Input: temp-threshold.
  - Processing: every observation the nodes confront his local value of the size of the distribution with the temp-threshold given by th coordinator plus the value of the behavior at the coordinator site, if is bigger a signal of up is sent to the coordinator, if the size is lower than the coordinator behavior a signal of down is sent.
  - Output: ack from the coordinator.
- Interact with the coordinator on the threshold managing: If the node is operating in a mix model with threshold algorithm and Holt-Winter model used alternatively the communicator send to the nodes the order to use one or the other model and request for the parameters and initial values for each algorithm.
  - Input: model to use and a request of initial parameters.

        - Processing: every observation the nodes confront his local value of the size of the distribution with the temp-threshold given by th coordinator plus the value of the behavior at the coordinator site, if is bigger a signal of up is sent to the coordinator, if the size is lower than the coordinator behavior a signal of down is sent.
        - Output: ack from the coordinator

- Coordinator:
    - Receive from nodes the updates of the stream, the *S(Q)*, the parameters for the correct forecasting.
        - Input: various kind of data for forecasting models.
        - Processing: Replace the existent data with the new given by the nodes.
        - Output: confirmation of update.
    - Send to nodes the values of the new temp-threshold: every time the number of the total updates of a given temp-threshold reach the number of node is necessary the update of this threshold.
        - Input: number of nodes, number of updates.
        - Processing: computation of new threshold.
        - Output: confirmation of update.
    - Interact with the Forecast Engine for get a correct behavior of the values at nodes site: the coordinator compute a forecast of the current value observed by the node x .
        - Input: number of the node, current time, smoothing, trend and seasonality arrays.
        - Processing: computation of the new values by the Forecast Engine.
        - Output: the predicted values.
    - Manage the threshold problem, and when necessary send to nodes the signals for switching from an algorithm to another: when the sum of the updates received form the nodes reach the threshold, or when the tracked valued above the threshold goes down this limit the coordinator have to send to the nodes a signal for change the tracking algorithm.
        - Input: threshold value, predicted N_t

    - Processing: send a signal of change algorithm.
    - Output: ack from the nodes.

- Frequency Vector:
  - Search and insert: for insert a value inside a frequency vector is necessary to implement sort and search.
    - Input: Frequency Vector and a value to search or insert.
    - Processing: transform the Frequency vector in an array, make a research for the value inside the array, and return the index of the value, with the index is easy to insert the value in the Frequency Vector.
    - Output: confirmation of insertion in the Frequency Vector
  - Sort: an algorithm for sorting the Frequency Vector is necessary for good performance in the value insertion and search.
    - Input: Frequency Vector.
    - Processing: a quick sort algorithm is executed on the Frequency Vector.
    - Output: sorted Frequency Vector.
  - Clear: for save memory at the node site a clear procedure for the Frequency vector is necessary, for the reutilization of these vectors.
    - Input: Frequency Vector.
    - Processing: Set all the values to zero.
    - Output: cleared Frequency Vector.
  - Sum or subtract two Frequency Vector: these operation are necessary to maintain a sliding window Frequency Vector
    - Input: two Frequency Vector.
    - Processing: Scan one vector, and add or subtract the multiplicity of the values selected to the correspondent value in the other vector.
    - Output: Frequency Vector.
  - Transform the Frequency Vector in an array: this operation is necessary to the functions of sort, search and ranking
    - Input: two Frequency Vector.
    - Processing: Create an array of the length of the Frequency Vector and fill it with the values gotten by the frequency vector.

        - Output: sorted array of Frequency Vector's values.

- Forecasting Engine:
    - Use different algorithm for forecast values.
        - Input: the number of the algorithm selected.
        - Processing: set as default forecast algorithm the one selected.
        - Output: confirmation of the selection.
    - Calculate all the values that are necessary to the forecast: this include not only the forecasted value but also the smoothing, trend and seasonality values that depend form the forecasted values.
        - Input: the current time, smoothing, trend, and seasonality arrays.
        - Processing: calculate the forecast value with the selected algorithm, and update the parameters arrays.
        - Output: forecasted value and updates parameters arrays.

4.2 UML

Unified Modeling Language™ (UML) is an industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems standardized by the Object Management Group[33][34]. UML simplifies the complex process of software design by using “blueprints” for software construction.

4.2.1 Use Case Diagram

A Use Case Model describes the proposed functionality of a new system. A Use Case represents a discrete unit of interaction between a user, human or machine, and the system. Each Use Case describes the functionality to be built in the proposed system, which can include another Use Case's functionality or extend another Use Case with its own behavior [34].

- Actors Diagram

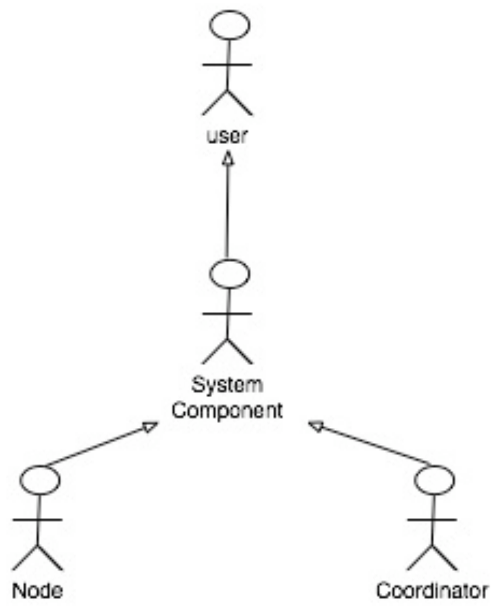

Figure 4.1: Actor Diagram.

The actors of this systems are not people, but are electronic device, because this is a simulator and not a program thought to be used by some one. These actors can do things, and interact each other inside the system.

- Global use case diagram

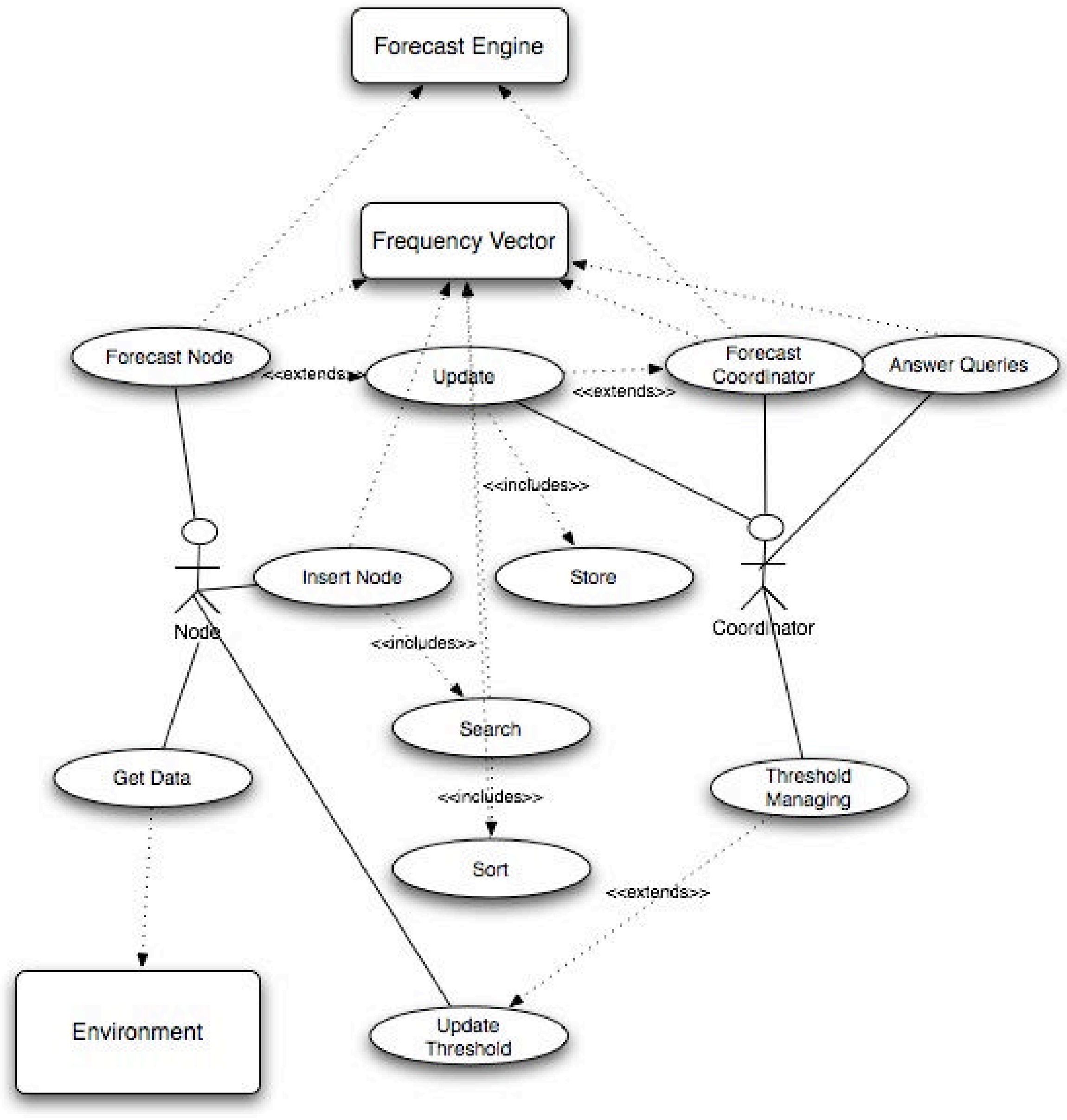

Figure 4.2: Global use case

In the figure above is shown the global use case diagram, that explains how functions are used by an actor to achieve a particular goal. This global use case is simple so I do not need to do specific use case for every actor. When the system is up the two actors have interactions, that represent communications, only in particular conditions when an error in the coordinator's or nodes' behavior is detected. The rectangular box represent the classes used by the functions for complete the procedure.

4.2.2 Activity Diagram

- Insert a new observed value

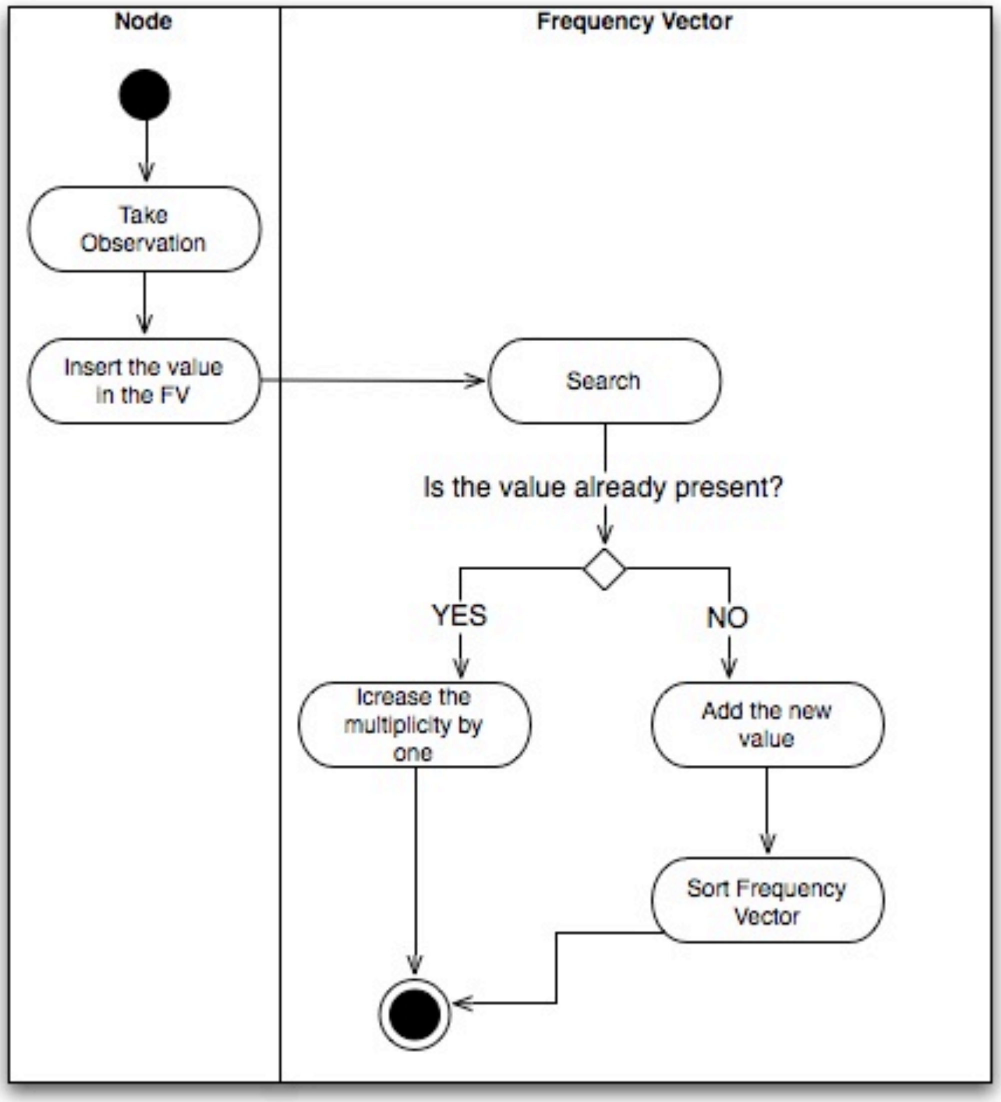
Node
Frequency Vector
Take Observation
Insert the value in the FV
Search
Is the value already present?
YES
NO
Icrease the multiplicity by one
Add the new value
Sort Frequency Vector

Figure 4.3: Insert values

The diagram above shows the actions that a node makes when receive or observe a new value. If the value is already present only the multiplicity of that value in the Frequency Vector is increased, although the node insert the new value at the end of the frequency vector, and then the entire vector is sorted. The cost of insert a new value in the right place or insert it at the and then sort the Frequency vector I s the same, but is easier the last manner because we can use easily the sorted method of the arrays.

- Update coordinator behavior

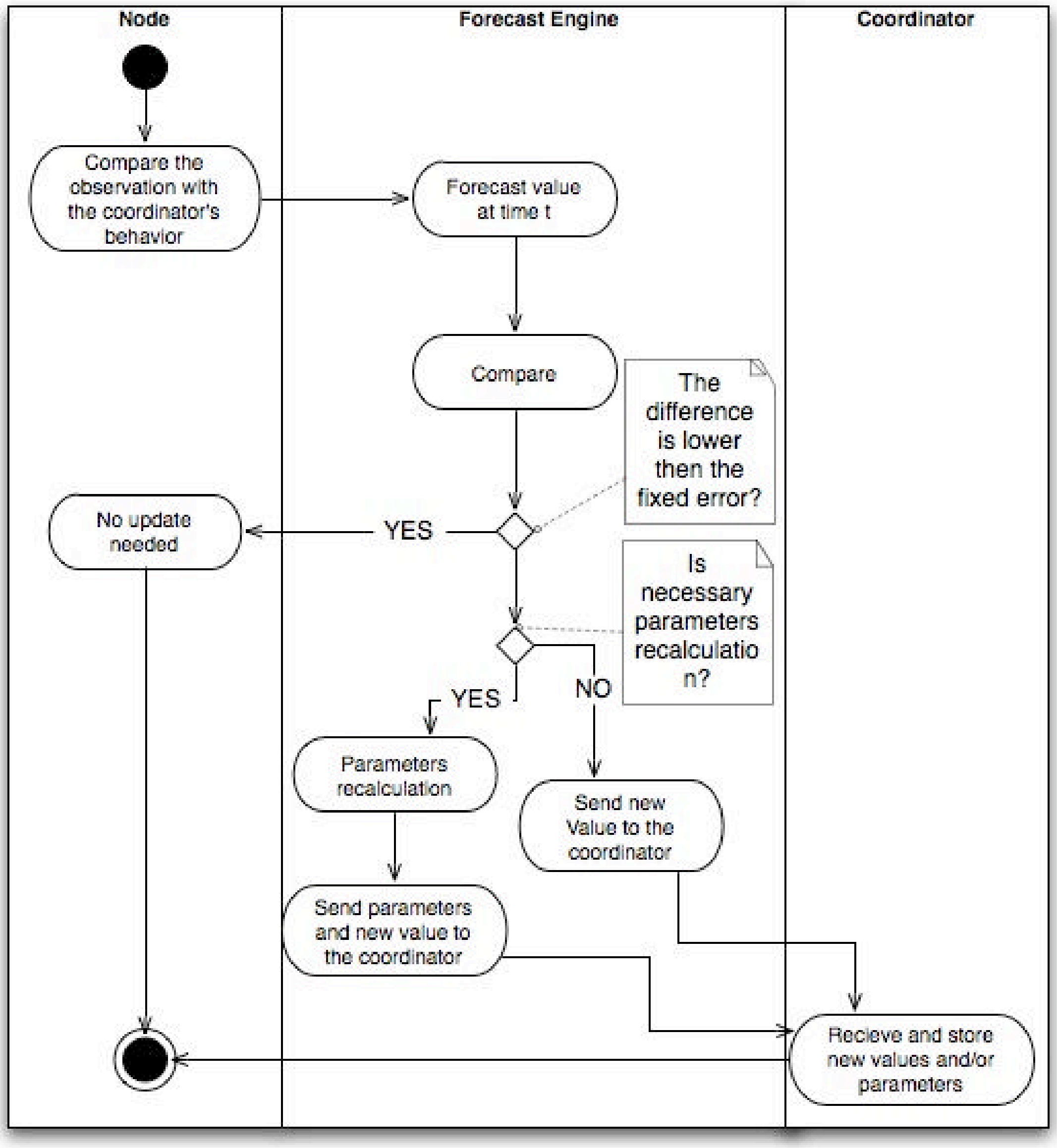
Node
Forecast Engine
Coordinator
Compare the observation with the coordinator's behavior
Forecast value at time t
Compare
The difference is lower then the fixed error?
YES
No update needed
Is necessary parameters recalculation?
YES
NO
Parameters recalculation
Send new Value to the coordinator
Send parameters and new value to the coordinator
Recieve and store new values and/or parameters

Figure 4.4: Forecast, node side

The nodes need to know the behavior of the coordinator for the value at the node site at time t. The node use the same model of the coordinator to predict the value at time t, if the value exceeds the fixed error an update is send to the coordinator. This is a simulator, so the coordinator send an acknowledge message to the node if it receives the update, in the real world no ack messages are exchanged from the two stations.

- Manage the Temp-Threshold inside the nodes

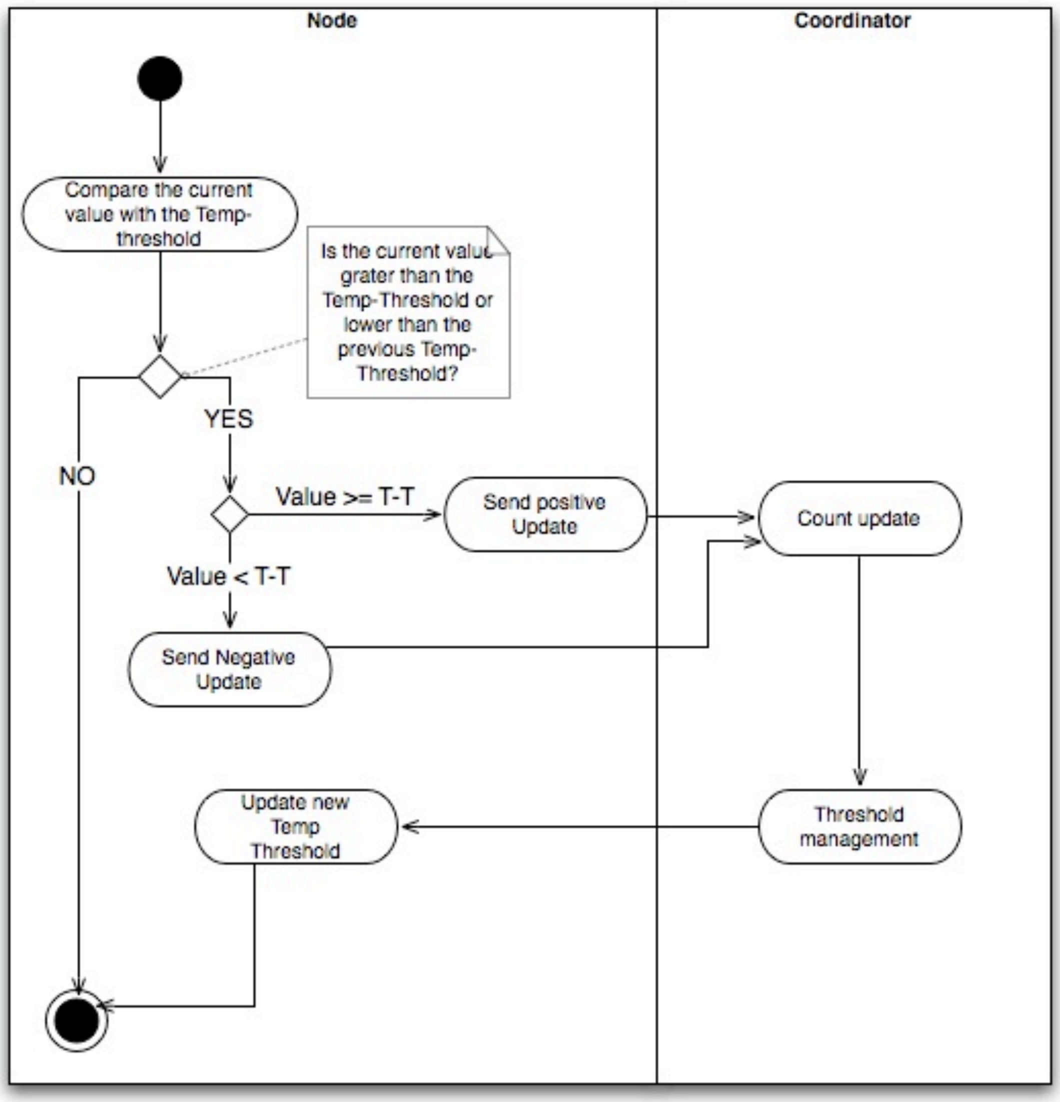
Node
Coordinator
Compare the current value with the Temp-threshold
Is the current value grater than the Temp-Threshold or lower than the previous Temp-Threshold?
YES
NO
Value >= T-T
Send positive Update
Count update
Value < T-T
Send Negative Update
Update new Temp Threshold
Threshold management

Figure 4.5: Node Temp-Threshold Managing

The figure above shows the activities that a node makes when check if the temp-threshold is broken by the current value. If it happens the node send an update signal to the coordinator, and receives always a feedback with the new temp-threshold value. The signal is a +1 or a -1 in this way the coordinator can easily sum the update values form the nodes and compare this sum in absolute value with the number of nodes.

- Manage Global and Temp Threshold

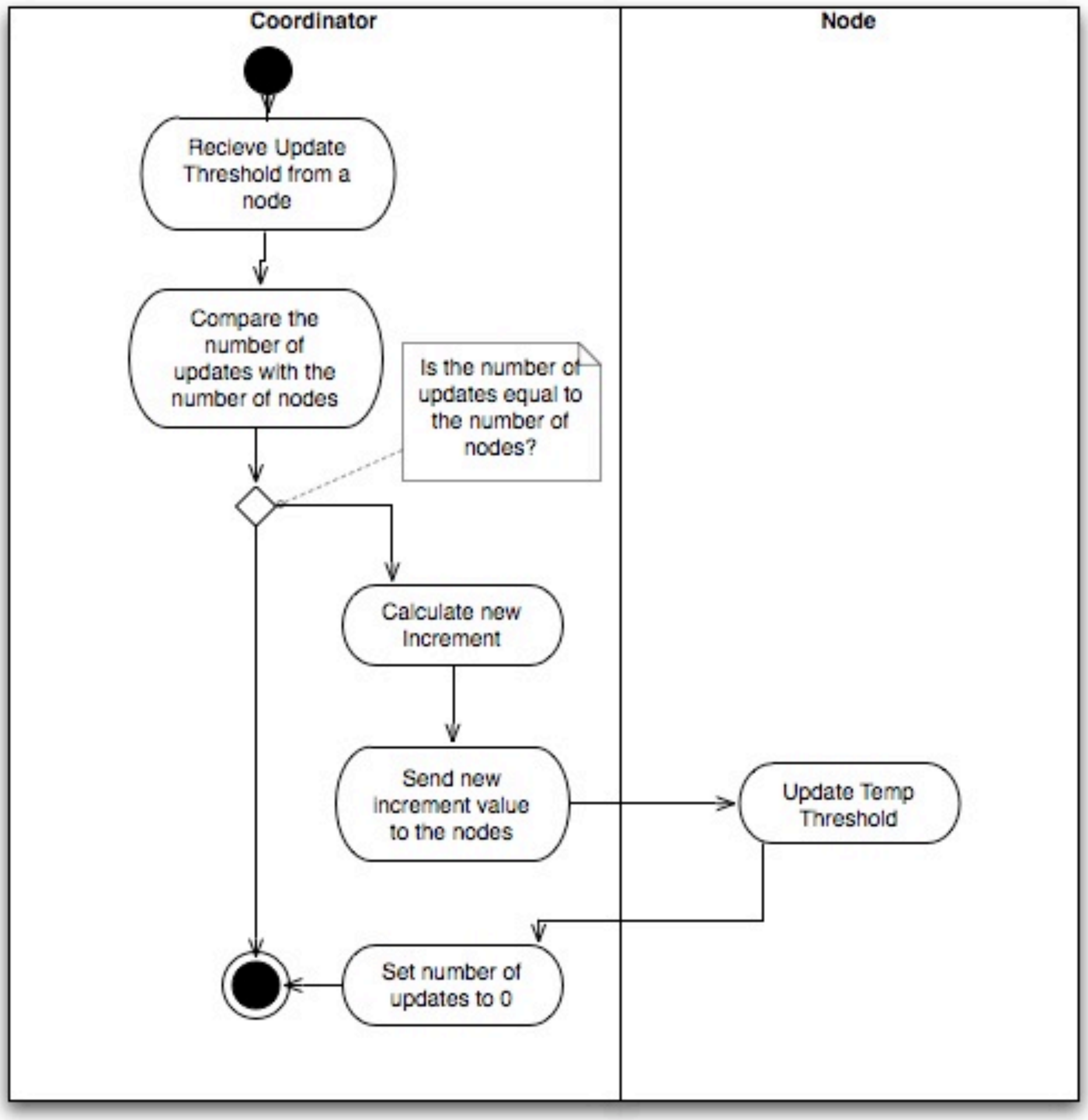
Coordinator
Node
Recieve Update Threshold from a node
Compare the number of updates with the number of nodes
Is the number of updates equal to the number of nodes?
Calculate new Increment
Send new increment value to the nodes
Update Temp Threshold
Set number of updates to 0

Figure 4.6: Coordinator Threshold Managing

One of the main coordinator activities is to manage the global threshold assign to each node a temp-threshold. For making this the coordinator receive signals from nodes when they broke the temp-threshold or go below the old temp threshold. If the absolute value of the update number is equal to the node number a new temp threshold is required, but is the value is negative, the threshold is equal to the previous threshold. If the value is positive a new Temp-Threshold value is calculated with the Yi algorithm. The coordinator sends the new value to the nodes, and set the number of signals received form the nodes to zero.

- Coordinator insert a new values at time t

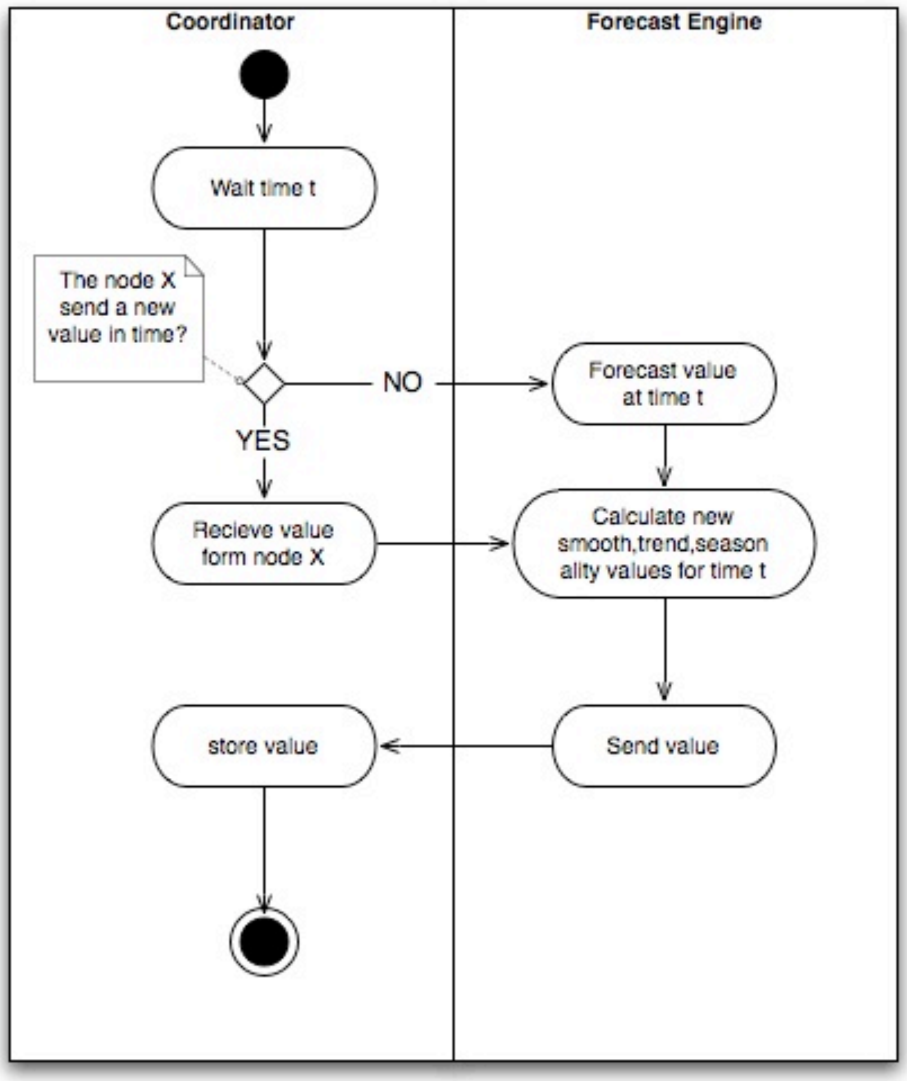
Coordinator
Forecast Engine
Wait time t
The node X send a new value in time?
NO
YES
Forecast value at time t
Recieve value form node X
Calculate new smooth,trend,season ality values for time t
store value
Send value

Figure 4.7: Insert at coordinator site

Every instant of a time step or window value the coordinator has to insert a new value in is behavior for the node *x*. If the node x does not send to a value to the coordinator in time, this means that the behavior of the coordinator is correct, so it can make a forecast of the next value and assume it as correct. If the node *x* send the update, the new coordinator store the new value and also the parameters if the node sent them together .

- Insertion of a value at coordinator site

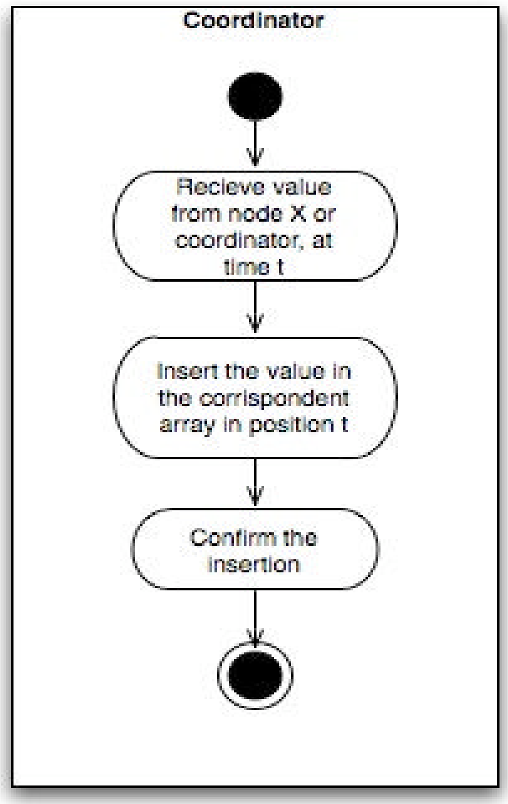
Coordinator
Recieve value from node X or coordinator, at time t
Insert the value in the corrispondent array in position t
Confirm the insertion

Figure 4.8: Insertion, Coordinator site

Every step time or window's element the N_j value has to be inserted in the correspondent array. This operation is very simple, and the coordinator does it every time period for every node, the value to insert could came from the node, if the behavior of the coordinator is incorrect, or from the coordinator that call the forecast engine for predict the next N_j.

- Search inside a Frequency-Vector

  The figure 4.9 shows he activity diagram of the search function. The search is one of the principal activity in this system, because is called every time an insertion is need in a node, so every time an observation arrive in a node, a search is made, and sometime also a sort is made. This happens because for the use of the binary search the array must be sorted for get the exact result, if is not sorted the result is a negative values. A frequency vector is composed by two vectors one with the values and one with the multiplicity of the values, for the search the system need only to transform the first vector in an array and then make a search inside it.

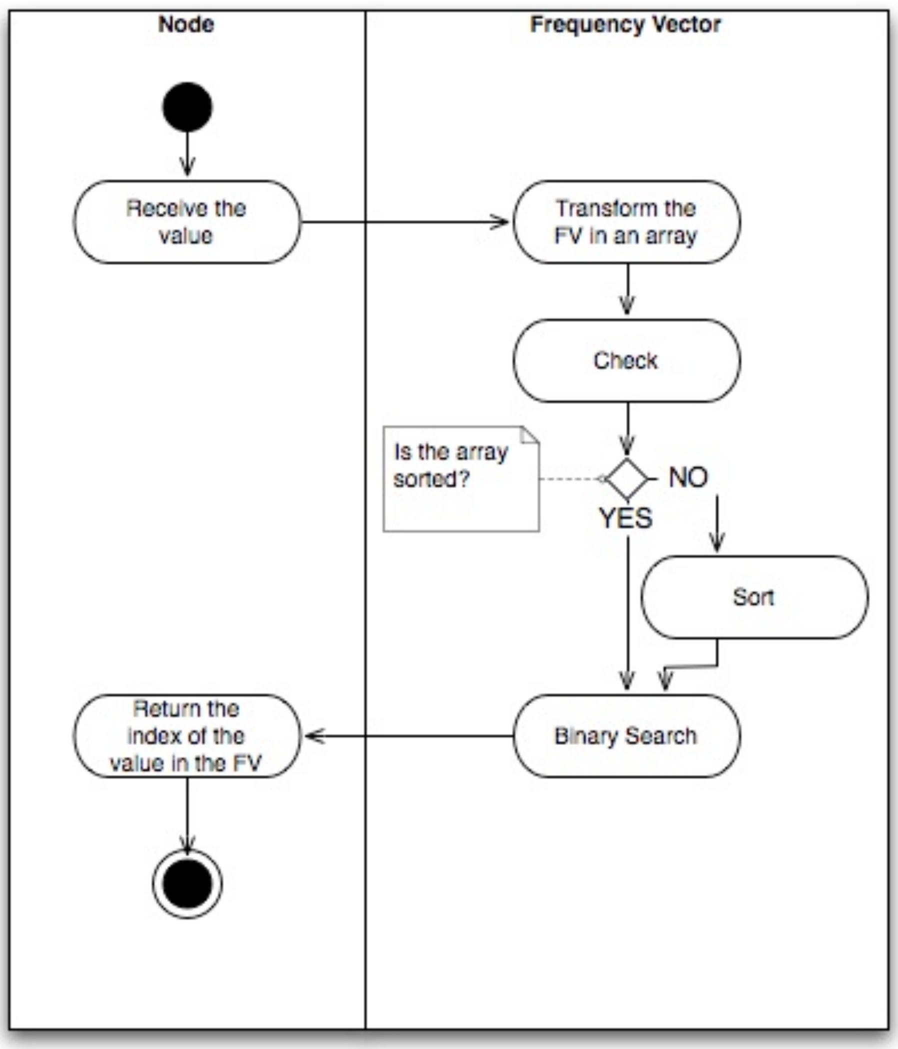

Figure 4.9: Search

- Sort a Frequency-Vector

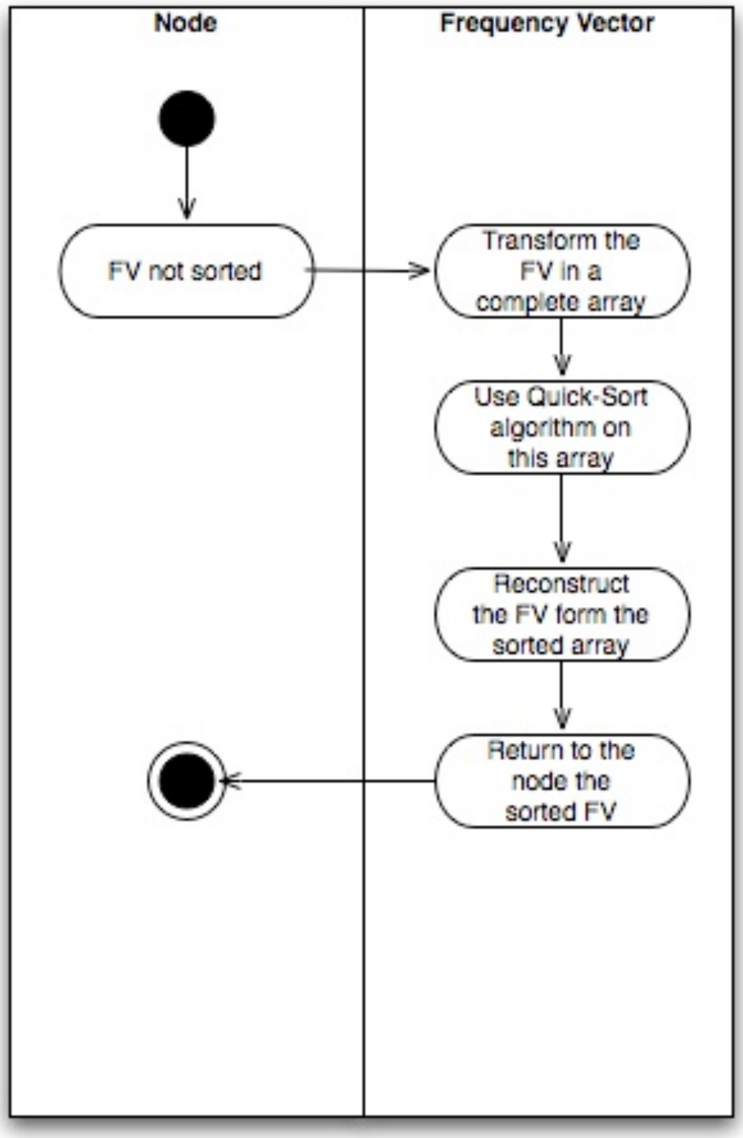
Node
Frequency Vector
FV not sorted
Transform the FV in a complete array
Use Quick-Sort algorithm on this array
Reconstruct the FV form the sorted array
Return to the node the sorted FV

Figure 4.10: Sort

The figure above represent the activity diagram of the function sort. This is a very important function because permit to insert a value correctly inside a Frequency Vector and permit a correct research. The Frequency Vector is expanded in an array, every value is repeated a number of times equal to its multiplicity, and then the array is sorted with quick-sort. From this array the system can in an easy way recreate the sorted Frequency Vector.

- Take an observation from a file

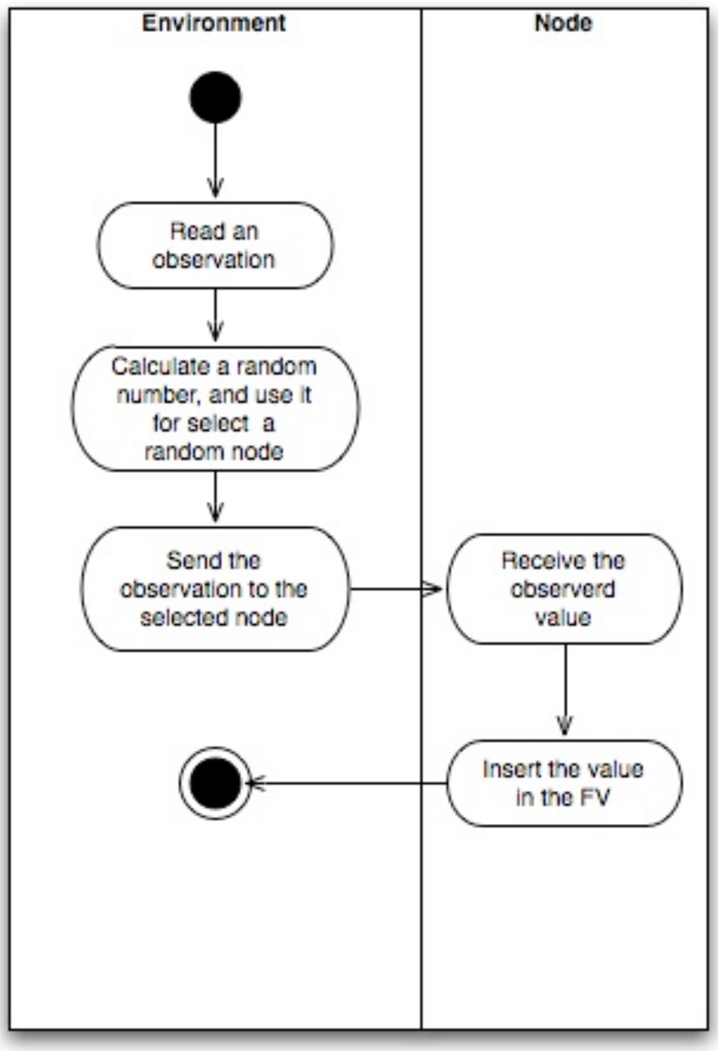
Environment
Node
Read an observation
Calculate a random number, and use it for select a random node
Send the observation to the selected node
Receive the observerd value
Insert the value in the FV

Figure 4.11: Observation

The value are taken by the environment form a data file, and then the environment make a random selection of the destination node. But the selection is not truly random, this is due to the algorithm use that not permit a real random selection, so in a long time period it can be recognized a pattern in the node selection and so in the nodes' values.

4.2.3 Class Diagram

The Class Diagram shows the classes and their associations. It has something in common with an entity-relationship diagram, it describes the static or structural parts of the system and the relationships between them. I will describe first every class in the simulator and the iteration that happen between those classes. In the description I will write only the most important variables and functions for each class because the variable and the function are too many for a complete explanation[34][35].

- EnvData

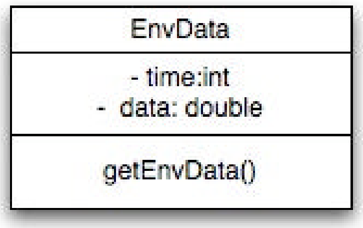

Figure 4.12: EnvData Class

This class represent the value observed. This value is composed by a variable time, that represent time in second elapsed from the beginning of the observations, this time is taken form the timestamps of the data. The second variable is data, that is the real value of the observation. The function getEnvData return those two variables.

- Environment

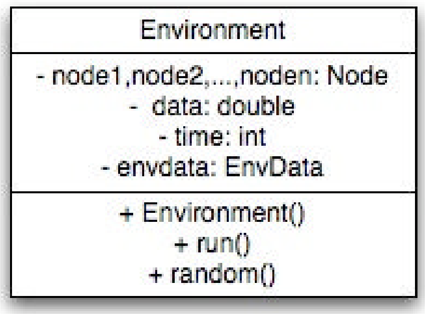

Figure 4.12: Environment Class

The Environment class, is the base class that push the observation to the nodes. This represent more an network environment than a nature environment. In network the data arrives to the server, are in other word pushed to them. In the natural world a temperature sensor pull a data from the environment making an observation. The variables are the nodes to use, an EnvData variable to send to the nodes and two temp variables to use to construct the EnvData variable. There is the normal constructor of the class, the input parameter is the number of the nodes and the name of the data file, there is a function run() because this class implements the threads, because I want to make possible to create different environment in different thread that could coexist together.

- Forecast Engine

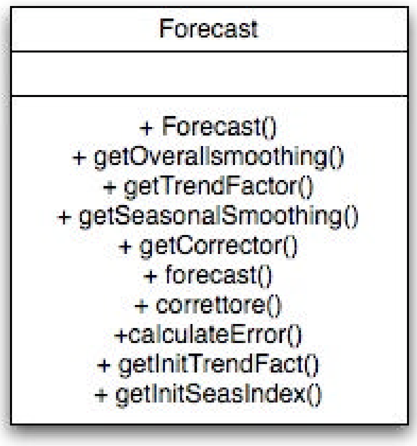

Figure 4.13: Forecast Class

This class is the real core of the simulator, and contains all the functions that are necessary for the correct forecast of the values at both, coordinator and node site. The forecast function calculate a forecast value with the selected algorithm, the other are auxiliaries functions that allow the prediction, for example, the getInitTrendFact function calculate the initial value of the trend factor, this class take in input the first two period series of values.

- Frequency Vector

This class give the data structure for the quantile calculation, and for the total count problem. Contains a variable Vector, that not require a fixed length in the initial declaration as array does. Inside at this variable I store an array of length two of double, in this array stays the value and the multiplicity of this value. The class provide some functions for operations between Frequency Vectors, and all the functions for the extraction of quantiles or for count the number of values inside a frequency vector.

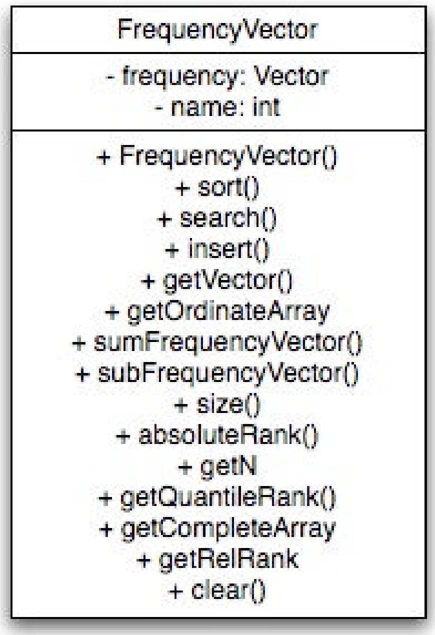

Figure 4.14: FrequencyVector Class

- SystemComponent

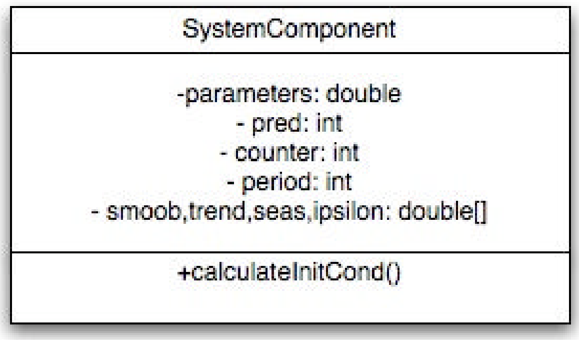

Figure 4.15 SystemComponent Class

This class is the base class for the two system component: nodes and coordinator. This two components basically have a lot in common and in the real world could be that a node can make the coordinator and vice-versa. But in this simulator I don't want to introduce routing and exchange problems, so I consider that a node can't change is state to coordinator. The variables are generic and include all the parameters alfa, beta, gamma, ro, the counter and the period of the data observed.

- Node

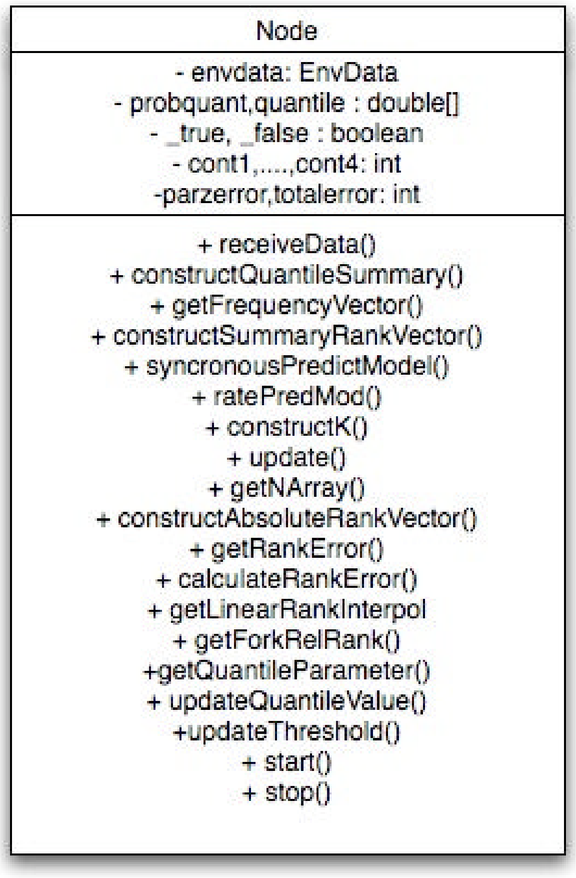

Figure 4.14: Node Class

The figure above shows the structure of the node class. The node is one of the principal elements, together with the coordinator, of the simulator. The class node inherit all the characteristics of the class SystemComponent, and adds new variables and function that make more specialized the class. The function receiveData() allows the class to receive EnvData form the environment, and the function update() call all the function needed every end of a time steps, or every slide of the windows. The others function are auxiliary function used for the calculation of the quantiles, the quantile ranks or the summary quantile vector.

- Coordinator

This class inherit all the function from the class SystemComponent and add all the functions necessary for the monitoring of all the nodes and for answer at the continuous monitor queries, so the coordinator know wit a ε-error at any time the value of N_j or the quantile summary. There are some characteristics function of this class such as fastForecast() that forecast the next value without insert the trend, seasonality and smoothing value in the relative arrays, this function is used for answer to the continuous-monitor queries in the time between two time steps or a slide of the window.

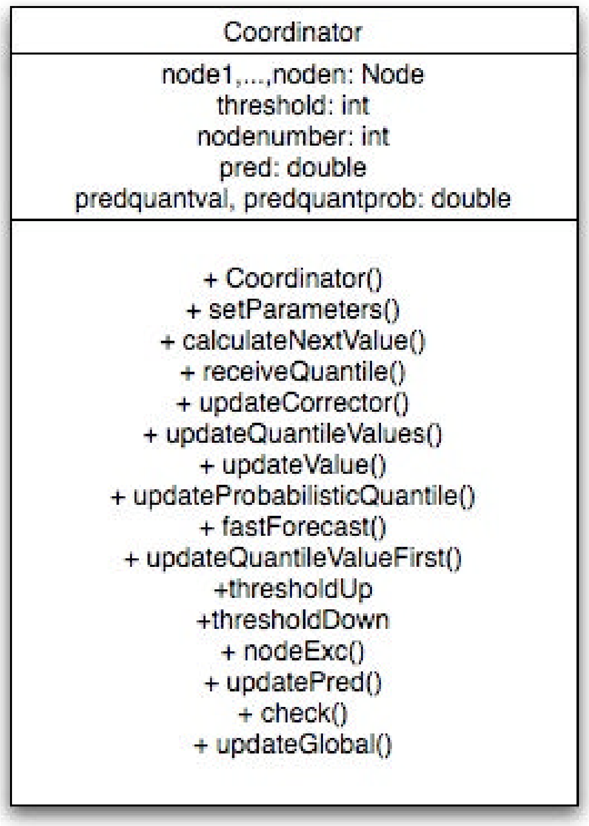

Figure 4.15: Coordinator Class

- Class association

  In the figure 4.16, are shown the class associations; is easy to see how the calluses interacts each others. The Environment could have al least one node, and every node has only one Environment from which its receives data, and only one coordinator. The Coordinator could coordinate one ore more nodes, and use only a forecast engine, as the nodes do. Node and Coordinator are derived from SystemComponent.

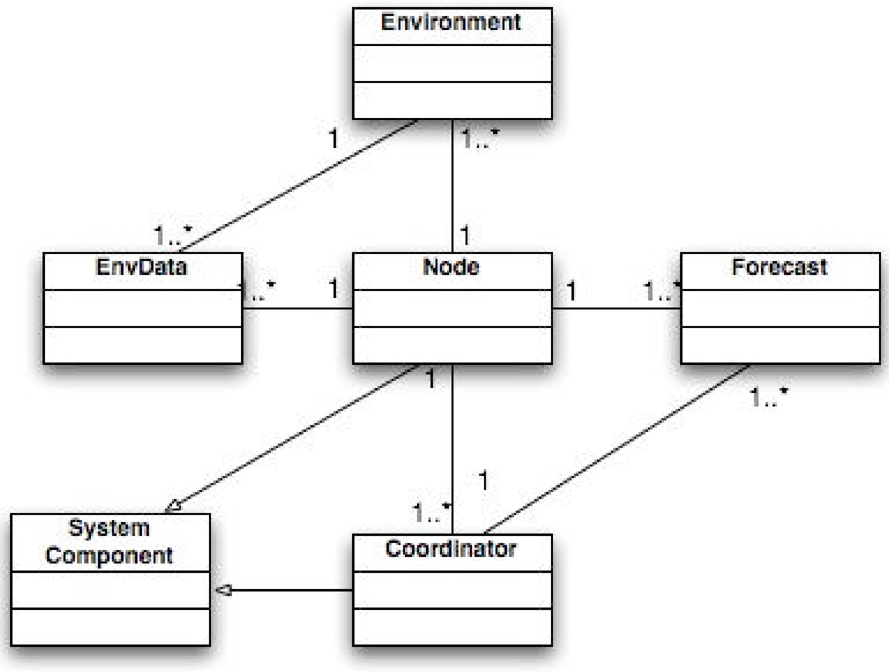

Figure 4.16: Class Associations and Implementation Inheritance

4.2.3 Sequence diagrams

The well-known Message Sequence Chart technique has been incorporated into the Unified Modeling Language (UML) diagram under the name of Sequence Diagram [34]. A sequence diagram shows, as parallel vertical lines, different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner [33]. The complete series of diagrams is too long for an exhaustive explanations in this thesis, so I will show only few interesting sequence diagrams.

- Coordinator update

  The diagram in figure 4.17 describe what happen when a time step end or the window slide, and the behavior of the coordinator is wrong. In this diagram I choose to show only the function for the total count monitoring, so no threshold and quantile monitoring.

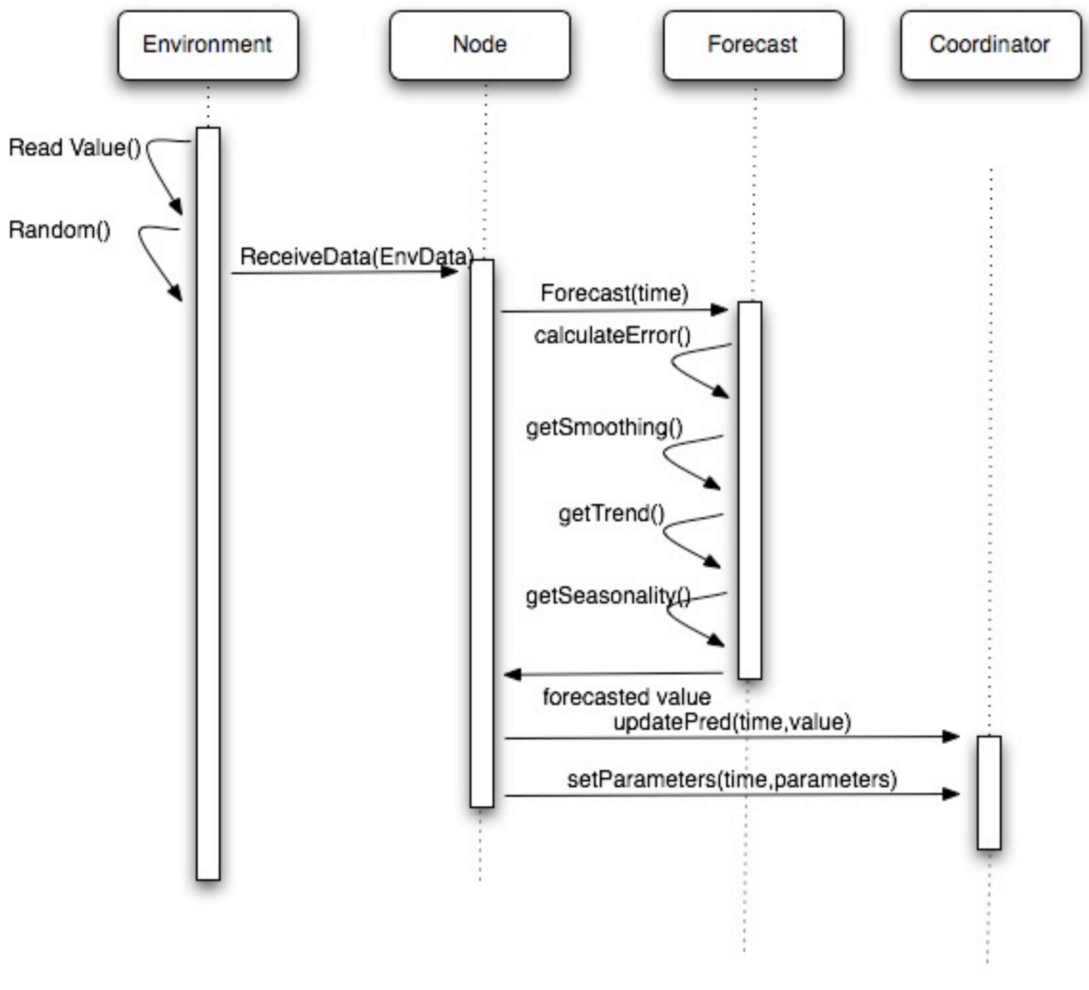

Figure 4.17: Coordinator update, sequence diagram

The Environment read the values from the data file the values and invoke a random function for the selection of the node. Every time the system forecast the value, it compare the observed value and the predicted value, if error is grater than the maximum tolerance, the parameters for the next forecast , such as trend, seasonality and smoothing are calculated using the observed data. In this diagram I simulated also that new parameters (alfa, beta, gamma, ro) are requested for the correct forecast at the coordinator site.

- Observation Insertion

  In the graph above is shown a very specific situation that happen every time period, but I omitted to express how happen every observation that is sent from the Environment to the node. The figure 4.18 express the sequence diagram of a generic insertion of an observation in the node's frequency vector. The data is received ad the inserted in the node using the function provided by the FrequencyVector class, The function insert, calls the function search, and this calls eventually the sort function.

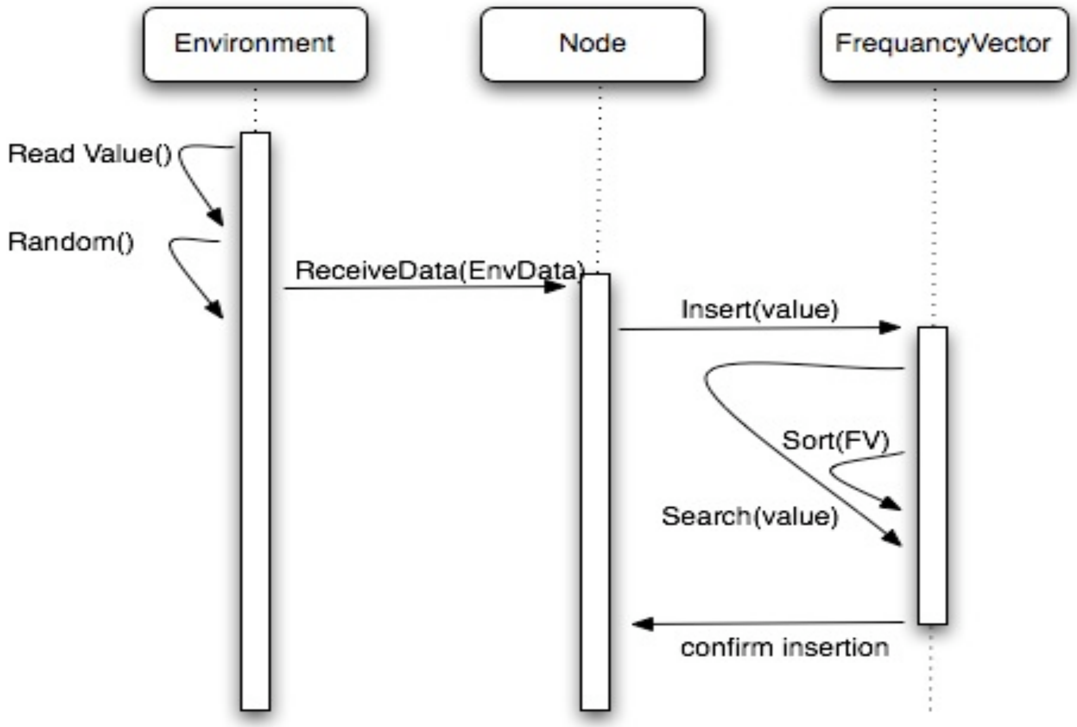

Figure 4.18: Node Insertion, Sequence Diagram

- Quantiles Generation

  Every time steps or every window slide, the node have to recalculate the quantile values and rank values for compare those values with the ones that represent the coordinator behavior for that node. The construction those arrays use some of the Frequency Vector's function. In particular constructK, that returns the number of element of the summary determined by the fi value used in the system. Get the complete array form the frequency vector is necessary for the correct construction of the summary that can include mirrored values. The rank summary is build making a number of calls to the rank function of Frequency vector equal to the number of the element of the summary. Once completed this quantile summary the node have to compare with the forecasted one and then eventually send new summary to the coordinator.

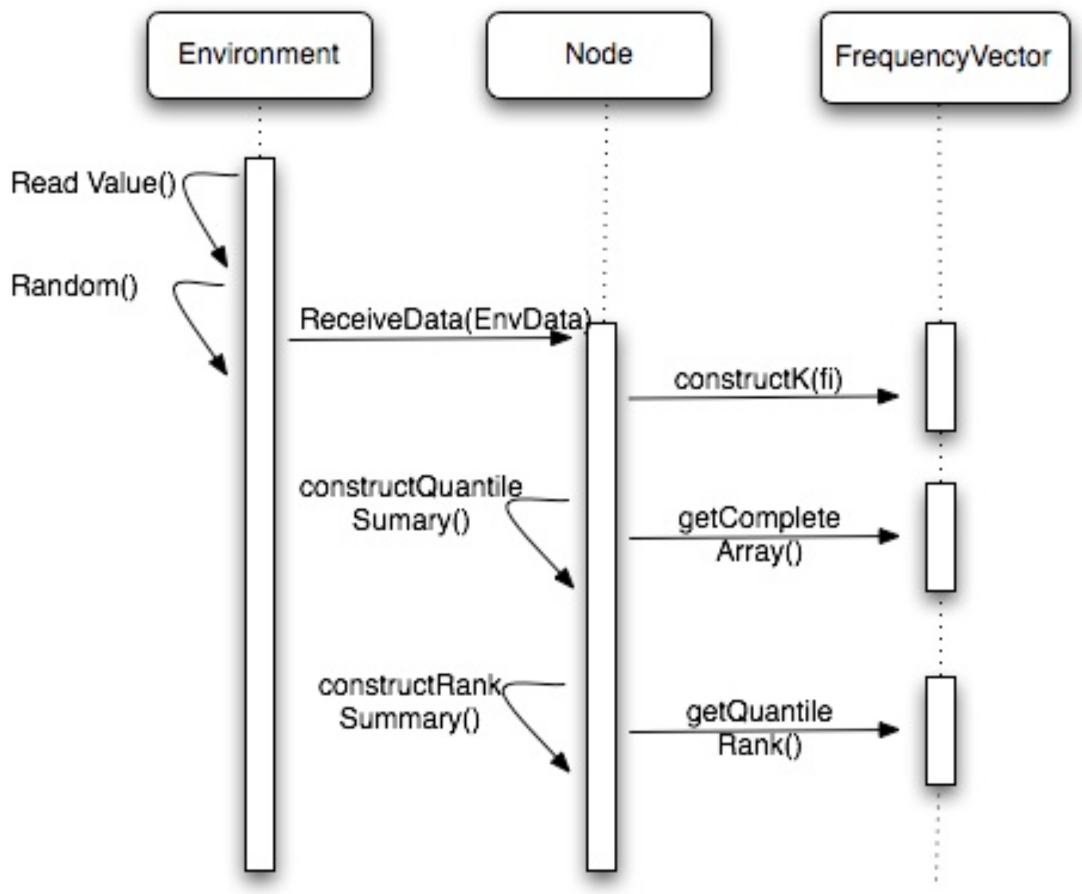

Figure 4.19: Quantiles Generation, Sequence Diagram

- Threshold Management

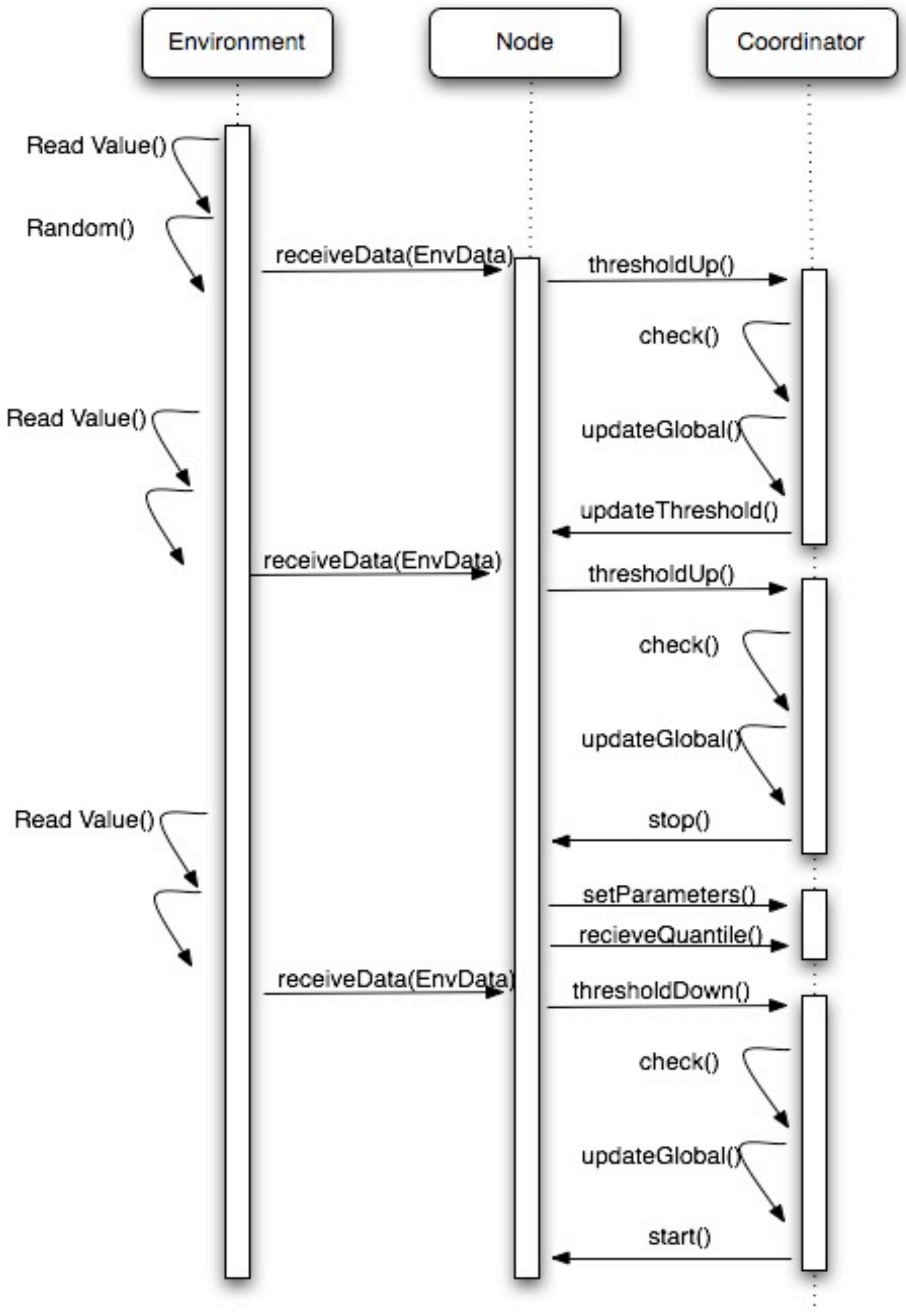

Figure 4.20: Threshold Management, Sequence Diagram

The figure 4.20 explain in a clear way hot the threshold management works, both node side and coordinator side. Make as assumption that all the observations represented in the diagram are observations at the time steps or at the window slide, I do not report all the observation because I was focused only on the thresholds. At the first time steps the node broke the temp- threshold value and send signal of thresholdUp to the coordinator. The coordinator first check if the number of updates in absolute value is bigger then the number of nodes (and in this case it is) and the recomputed a new temp threshold and the resend them to all the nodes. At the next time steps the new temp threshold is broken another time and the node resend a thresholdUp() to the coordinator , this time I want to show what happen when the total value is near ε to the global threshold value. The coordinator send a stop signal to all the nodes, and now the nodes switch from the threshold monitor to the quantile tracking monitor, sending at the coordinator all the values that the coordinator needs for forecasting the nodes' observations. At the last time step shown the value goes down the last temp threshold that was given to the node, so the node sent to the coordinator a thresholdDown signal. The coordinator at this point check for the number of updates, and if the number is equal to the number of nodes the coordinator send a tart signal to all the nodes with the new threshold and switch form the quantile-monitoring to the threshold monitoring.

4.3 Implementation

I choose to write this simulator using Java language, for two main motivations: first I need an multiplatform language, because I used Mac OS X for the develop of the simulator and for the test I used a Linux Fedora machine provided by the AT&T, second because I find a lot of useful classes in java, on for all is the Vector class that provide a more powerful version of the normal List Class.

The datasets used are in text format with some separators, and I do not use external program for the parsing of the data, all is made inside the Environment Class, that provide data to the nodes. The only problematic thing was the random number generatior, provided by the Java Math class, is not truly random and respect a sort of path for a lot of generations, I generate more than twenty millions of random numbers. If the algorithm decides at the beginning that the node A receives a 22% of the updates this values can change during the time but remains stable around the 22%. But this is not necessary a bad thing because this respect the real computer network environment with load balancing server, In the modern datacenter the traffic that arrives to the server is not random but is send to that server because there is a path for determine to which server send the update. So this random generator respect the distribution of the data inside

5. CONCLUSIONS

All the work of optimization that I done is really necessary inside a sensor networks because the communications between nodes and coordinator caused by answering the continuous-monitoring queries introduce unnecessary overhead in the network; the problem is harder in wireless sensor networks because more communication means more use of energy and less battery life For example, in a sensor network composed by a router, the value to be monitored usually is the number of connection or the number and the size of connections (as for the Word cup '98 dataset) and if no optimization is performed in query execution, as much connections arrive to the routers as much the overhead caused by the monitoring increase. The nodes in the wireless sensor networks are usually powered by batteries; to make a wireless communication is very expensive in terms of power consumption, so optimizing the continuous-monitor queries in the WSN is a must. The results of my tests with the simulator, using my new algorithms for the continuous-monitoring show a good reduction in the number of communications, this means less overhead and less battery consumption.

I divided the main continuous-tracking problem in its three principal monitoring sub-problem:

- Total-Count
- Quantile
- Threshold

The first and the second problems are strictly correlated each other, a good result for the first problem permits a good result for the second problem. When is obteined optimization of the continuous monitoring queries involving the quantiles the following goal is to avoid unnecessary communications between nodes and coordinator, introducing an approximation error ε in the queries. With this error the coordinator guarantee ε-approximate queries' answer. A communication between a node and a coordinator is made only when the coordinator's behavior is wrong in excess of ε. So a good model for forecast the nodes' behavior is a good way for optimizing communications and thus continuous-monitoring queries inside a sensor networks. I find that the sequence of the number of the updates received from nodes in the time (total count) could be seen as a time series. I begin analyzing the typical forecasting algorithms used in time series and I chose the Holt-Winter as base algorithm and than modified it with a new error correction addendum for achieving better performance than the unmodified. I also use a different method for the parameters selection based on the minimization of the error made by the algorithm and not on the minimization of the distance between the forecasted time series and the observed time series. Using this new algorithm I could get 41% less error (in average) using the sliding windows time division and the 25% less wrong values using the time steps.

This new algorithm allowed me to get a good result in the quantile tracking, because its based on N_j, so if we have that the behavior of the coordinator for the number of updates stay inside the bounds fixed, there is a very high probability that also the answers to the quantile monitoring queries will be inside the ε fixed bound. The basic formula for the quantile tracking in explained in the Cormode et al.'s papers, but could be improved for getting better result. I used different algorithms to calculate the predicted rank, and a different way to calculate the rank summary at the coordinator site. This two changes permitted to have the predicted rank at the coordinator site nearer the real rank for that value, and so to improve the performance of the whole system. The tests made with the simulator showed 70% less communication involving the quantile summary using my new two method together with, respect to the original Cormode et al. tracking algorithm. The tests also show an average 10% reduction of the number of communications that include only the N_j by using my algorithms, this could happen thanks to the new algorithm for the calculation of the predicted rank value, this good algorithm permits in most of the cases to update only the N_j value instead of the whole quantile summary.

The last step of the optimization of continuous-monitoring queries involved the monitoring of the threshold. In most of the cases we don't want to monitor a value if the value is under a fixed threshold because the values are not interesting for our purpose. So for optimizing the continuous-monitoring problem is necessary to find a good algorithm for the distributed monitoring of the threshold. As the environment typical of a sensor network is distributed and the nodes could have very different distributions, the goal is to provide to the coordinator a good behavior of the distribution and so of the total-count value in the nodes. To obtain that I modified the Yi algorithm for threshold tracking, in order to utilize it in a sliding window environment, and not only in an incremental environment as in origin. This algorithm permitted to raise an alarm when the total number of updates is near ε to the threshold. It is very useful for the continuous monitoring, so the continuous quantile tracking can start only if the values are really interesting, and not if the values are irrelevant. This combination of the threshold tracking and the quantile tracking optimized in the sensor networks all the queries like: monitor this value, with ε-approximation, only when the value is above x.

The threshold problem in general is open, my solution is not definitive, because it does not solve the situations where the value to monitor oscillates under and over the threshold or the temp-threshold generating a lot of communications between the nodes and the coordinator.

REFERENCES

[1] Römer, Kay; Friedemann Mattern (December 2004). "The Design Space of Wireless Sensor Networks". IEEE Wireless Communications 11 (6): 54-61.

[2] Thomas Haenselmann (2006-04-05). "Sensornetworks". GFDL Wireless Sensor Network textbook. Retrieved on 2006-08-29.

[3] Hadim, Salem; Nader Mohamed (2006). "Middleware Challenges and Approaches for Wireless Sensor Networks". IEEE Distributed Systems Online 7 (3). art. no. 0603-o3001.

[4] R. Beckwith, D. Teibel, and P. Bowen. Pervasive Computing and Proactive Agriculture. In *Adjunct Proc*. PERVASIVE 2004, Vienna, Austria, April 2004.

[5] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. "Continuously adaptive continuous queries over streams. In *proceedings of* ACM SIGMOD, 2002.

[6] Deshpande and M. Hellerstein. "Lifting the Burden of History from Adaptative Query Processing". In *proceedings of* VLDB, 2004

[7] G. Cormode and M. Garafalakis. "Sketching Streams Through the Net: Distributed Approximate Query Tracking". *In proceedings of* VLDB, 2006.

[8] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. "Gigascope: A Stream Database for Network Applications" . In *proceedings of* ACM SIGMOD, 2003.

[9] S. Madden, M. J. Franklin, J. M. Hellerstein, and H. Wei. "The Design of an Acquisitional Query Processor for Sensor Networks". In *proceedings of ACM SIGMOD, 2003.*

[10] Cormode, M. Garafalakis, S. Muthukrishna, and R. Rastogi. "Holistic Aggregates in a Networked World: Distributed Tracking of Approximate Quantiles" In *proceedings of* SIGMOD, 2005.

[11] B. Babcock and C. Olston. "Distributed Top-K Monitoring". In *proceedings of* ACM SIGMOD, 2003.

[12] C. Olston, C. Buragohain, D. Agrawal, and S. Suri. "Medians and beyond: New aggregation techniques for sensor networks". In *proceedings of* ACM SenSys, 2004.

[13] Das, S. Ganguly, S. R. Madden, J.M. Hellerstein, and W. Hong. "Model-Driven Data Acquisition in Sensor Networks". In *proceedings of* VLDB, 2004.

[14] Wikipedia [Online]. Available: http://en.wikipedia.com

[15] [A. C. Gilbert, y. Kotidis, S. Muthukrishna, and J.M. Strauss. "How to Summarize the Universe: Dynamic Maintenance of Quantiles" . In *proceedings of* VLDB, 2002.

[16] M.B. Greenwald, and S. Khanna. "Space-Efficient Online Computation of Quantile Summaries". In *proceedings of* ACM SIGMOD, 2001.

[17] M.B. Greenwald, and S. Khanna. "Power Conserving computation of Order Statistics over Sensor Networks" . In *proceedings of* ACM PODS, 2004.

[18] G. S. Manku, S. Rajagopalan, and B.G. Lindsey. "Approximate medians and other quantiles in one pass and with limited memory". In *proceedings of* ACM SIGMOD, 1998.

[19] Statsoft Textbook [Online]. Available: http://www.statsoft.com/

[20] Engineering Statistic Online]. Available: http://www.itl.nist.gov/div898/handbook/index.htm

[21] G.E.P. Box, G. Jenkins. Time Series Analysis: Forecasting and Control, 1976. (Holden-Day)

[22] Index of time series data set form Box, Jenkins, and Reinsel [Online]. Available: http://www.stat.wisc.edu/~reinsel/bjr-data/index.html.

[23] P. F. Velleman, and D.C. Hoaglin. Applications, Basics, and Computing of Exploratory Data Analysis, 1981. (Duxbury Press, Boston)

[24] Sir M. Kendall. Time-Series (2nd ed.). 1976 (Charles Griffin & co. Ltd., London)

[25] D. McDowall, R. McCleary, E.E. Meidinger, R.A. Hay. Interrupter time series analysis, Sage. Newbury Park, CA, 1980.

[26] Internet Traffic Archive [Online]. Available: http://ita.ee.lbl.gov/

[27] UC IPM [Online]. Available: http://www.ipm.ucdavis.edu/

[28] R.F. Nau, Introduction to Arima [Online]. Available http://www.duke.edu/~rnau/411arim.htm

[29] J.D. Brutlag. "Aberrant Behavior Detection int Time Series for Network Monitoring" *In proceeding* of USENIX 2000.

[30] F. Haro, M. Chhaparia, and L. Cottrell. Detecting Loss of Performance in Dinamic Bottleneck Capacity (DBCAP) Measurements using the Holt-Winter Algorithm, [Online].Available: http://www-iepm.slac.stanford.edu/monitoring/forecast/hw.html

[31] R. Kralapura, G. Cormode, J. Ramamirtham. Communication-Efficient Distributed Monitoring of Thresolded Counts

[32] K. Yi . Distributed Monitor of Frequency Moments.

[33] R. Miles, K. Hamilton. Learning UML 2.0, O'Reilly 2006

[34] M. Flower. UML Distilled, A Brief Guide To The Standard Object Modeling Language. Addison-Wesley, 2006.

[35] Holub Associates: UML References Card [Online]. Available: http://www.holub.com/goodies/uml/