

Cooperative AI
in an
Unbalanced Distributed Environment

Hans Joerg Prueller
hans.prueller@gmx.net

November 2005 - May 2006

Distributed Learning Investigation

Table of Contents

1 Cooperative, Distributed Learning.....	3
1.1 Introduction.....	3
1.1.1 Proposed System Architecture.....	3
1.1.2 Participating Roles, Relationships.....	4
1.2 Restricted Monitoring Node.....	4
1.2.1 Machine Learning on Restricted Hardware.....	4
1.2.1.1 Power Consumers.....	4
1.2.1.2 Power Consumption Figures.....	5
1.2.2 Conflicting aims of the restricted device.....	8
1.3 Computational Characteristics of Classifiers.....	10
1.4 Increasing Energy Efficiency of Classifiers.....	10
1.4.1 Existing/Related Research.....	10
1.4.2 Data Compression.....	10
1.4.3 Data Abstraction.....	11
1.4.4 Dynamic Sampling.....	12
1.4.5 Energy Scavenging.....	12
1.5 Backend Node.....	12
1.6 Local vs. Remote Processing.....	12
1.6.1 Related Research.....	13
1.6.2 Energy related issues.....	14
1.6.3 Further Issues.....	22
1.7 Knowledge Exchange.....	22
1.7.1 Knowledge Representation of Classifiers.....	23
1.7.2 Mutual Training of Classifiers.....	23
1.8 Communication / Data Transmission.....	23
2 Appendix.....	25
2.1 Schemes of the proposed monitor model.....	25
2.2 References.....	27

Distributed Learning Investigation

1 Cooperative, Distributed Learning

1.1 Introduction

Due to restricted processing resources, a computationally efficient learning algorithm has to be implemented on a restricted (in most cases) mobile node. As stated in [MIT97], machine learning can be seen as the search for the correct target function within a large hypothesis space. In the context of monitoring a persons health the hypotheses space is huge, the search with available resources on a mobile device for sure will take too long. It is clear that more sophisticated approaches have to be investigated to solve this problem. Additionally the restricted energy resources have to be considered.

Even though we're looking for a “simple” learning environment on the restricted mobile node, the decisions that have to be made are essential: if the mobile node falsely decides to have a dangerous situation and opens up the costly GPRS/UMTS line, battery power is consumed without a reason. If the mobile node decides to “wait” before raising an emergency alarm, the patients health could get into a dangerous state – in the most extreme case the patient could even die. A too pessimistic algorithm will exhaust the energy resources (and reduce the systems applicability), a too optimistic approach threatens the patients health.

Another point that has to be considered is how common learning algorithms acquire their knowledge: in most cases through exhaustive training experience. Learning algorithms improve with the increasing number of training datasets they observe. Monitoring a patients health does not allow to learn from training assumptions, a persons life depends on the system to correctly classify its health state. The learning algorithm can be “trained” by a set of pre-defined medical rules being representative for a broader population, but these rules can't be transferred to each patient unchanged. Every person is different, has a different lifestyle and different basic health conditions. The learning algorithm has to learn about the specific lifestyle and biological conditions of a person to correctly identify the persons health situation.

1.1.1 Proposed System Architecture

- describe architecture of the proposed system

1.1.2 Participating Roles, Relationships

- describe participants in system (mobile node, backend node, patient, external clinical personal)
- describe relationships between roles, e.g. backend node is supervisor of mobile node; patient is supervisor for both monitors, etc.
- human expert supervisor available at the backend node for all remote monitors

1.2 *Restricted Monitoring Node*

1.2.1 Machine Learning on Restricted Hardware

What are the specifics of restricted (in most cases mobile) machine learning compared to other machine learning systems (standalone, multi-agent systems, etc)? The key characteristics can be derived by the words “mobile” and/or “embedded” which can be interpreted from my point of view as:

- **small, robust, portable** and
- **connected** (to a backend system) with the need for
- a **long run-time** of battery and **minimal need for maintenance**

The above list also gives a short repetition of reasons why the mobile learning algorithm runs on a very limited device.

In the introduction of this document, I mentioned some similar research projects as “semi-mobile”. By comparing the attributes of those projects with the definition above clearly states that they are no real mobile monitoring systems (ie PDAs or Smartphones are not robust enough, have no long battery runtime and have regular need for maintenance).

1.2.1.1 *Power Consumers*

Which components/subsystems on the remote monitor device are consuming power? We have three major power consumer groups:

- Communication (the GSM/GPRS modem):

- setting up a connection
- tx/rx data
- idle or off mode
- Processing (CPU, memory)
 - idle
 - working
- Sensors (or communication with sensors via Bluetooth, ZigBee)
 - idle
 - reading data

As this study mainly focuses on the cooperation between mobile and backend node we will not focus on issues which are not relevant, i.e. The energy consumption used for reading sensor data is required in all cases: local, remote or else processing. Therefore this part will be treated “as given” but won't be investigated in more detail.

The relevant questions for this research start right when the sensor data is available at the processor: at this point in time the system has to decide what to do with the data: local or remote processing? Identify a normal or an emergency situation? Raise an alarm or not? Etc.

1.2.1.2 Power Consumption Figures

Power consumption characteristics of components of the restricted mobile monitoring device have to be investigated. Modern devices offer a “wireless CPU” instead of separate CPU and GSM/GPRS modem. Wireless CPU is a GSM/GRPS modem and a powerful CPU in one component (reference: wavecom wismo quik docs from andrew newton!). Most GSM/GPRS modems also offer “idle” or “off” modes where power consumption is 0 or minimal (<10mA).

Exemplary power consumption of GSM/GPRS modems:

- Advanced Wireless Planet GPRS Module (tri-band GSM)

Power consumption: Idle mode: <3.5 mA, speech mode: 250 mA (average)

<http://www.gsm-modem.de/gprs-module.html>

- ProComm Tri-Band GSM/GPRS MODEM

Power consumption: Idle Mode : <3ma Dedicated mode : 250ma

<http://www.cdl-products.com/ProComm.htm>

- GPS Vehicle Tracking System with GSM Modem

Power cons.: GPS Power down, GSM Power ON: 100 mA

GPS Power down, GSM standby: 5mA

http://www.ravirajtech.com/vehicle_tracking_system_india.html

- Sony Ericsson EDGE GPRS Modem (EE54)

Power cons: 350mA in EDGE comm, 5mA in idle mode

http://www.sonyericsson.com/downloads/EE54_R1B.pdf

- Sony Ericsson GPRS Modem (GM47)

Power cons: 350mA in GPRS comm, <5mA in idle mode

http://www.sonyericsson.com/downloads/GM47_GM48_R1L.pdf

- Siemens MC55/MC56 GPRS Modem

Power cons: 260mA in GSM 850/900

180mA in GSM 1800/1900

300mA in GPRS 850/900

230mA in GPRS 1800/1900

http://www.synergy-gps.com/Datasheet_MC55_56.pdf

- WISMO Quik Q2686 wireless cpu

power off: 25 microA (minimal power for clock needed)

Distributed Learning Investigation

in GSM/GPRS 1800/1900: averages

150-240mA for tx/rx

2.3mA in idle mode

avg GPRS C110 300-480 mA

(reference! - source from Mr. A. Newton, Linkwave)

- WISMO Quik Q2406, Q2426 wireless cpu

power off: 5-10 microA (minimal power for clock needed)

in GSM/GPRS 1800/1900: averages

150-235mA for tx/rx

2-6.5 mA in idle mode

avg GPRS C110 240-410 mA

(reference! - source from Mr. A. Newton, Linkwave)

Power consumption of processors:

- WISMO Quik Q24xx “wireless CPU” family

wireless CPU = integrated GSM modem and processor, if modem is unused it can be turned off

average consumption with turned off GSM:

8,5 – 9,5 mA (fast standby mode, open at max mode, network deregistration mode)

(reference! - source from Mr. A. Newton, Linkwave, power-modes pdf)

In addition, also power consumption of wireless communication with sensors (via Bluetooth, ZigBee) has to be considered. Like GSM/GPRS modems, also the short-range wireless

communication components consume 0 or only some micro-Amps in idle mode. When data is transmitted, about 10mA can be assumed (source: Eric – get some official documents to support this).

Data Transmission Power vs. Data Processing Power

As above figures clearly show, data transmission is much more costly than local data processing: GSM modems consume about 200mA in the average, embedded processors like the wavecom 24xx family “wireless CPUs” consume avg. 9mA for the processor. We can follow that **data transmission is about 20 times more power exhaustive than local data processing** – when the execution time is not considered.

The question of interest for a application optimizing local vs. remote processing would be: how long does it take to process the data locally (local power consumption * processing time) and how long does transmission of data and receiving of results take (local gsm modem consumption * communication time)?

This question provides the base for a more detailed analysis of “local vs. remote” processing optimization (see chapter 3.6).

1.2.2 Conflicting aims of the restricted device

Why can't just a simple classifier be used on the remote device to reduce energy consumption (e.g. fixed rule sets, if-then-else rules, decision matrix, etc.)?

Neglecting previously mentioned characteristics and already proposed system requirements and architectures – what is the *main* target of a general monitoring system on a restricted (mobile or embedded) device?

The **main target** of monitoring systems on restricted devices is to **reach a maximum autonomous system-run time by having minimal tolerance against missed alarms**. Or to be short, reach

- minimal energy consumption and
- maximum alarm-hit rate

Distributed Learning Investigation

Thinking further, maximizing the alarm-hit rate also implies maximizing (or at least increasing) data transmission to a backend system (as every alarm has to be sent somewhere): Maximizing alarm hit rate means that the system must not miss any alarm, in case of doubt an alarm has to be raised for safety. To avoid oversight of a single true alarm, the system has to take increased false alarms into account. This results in increased data transmission which, as a consequence, clearly constructs a conflict: Data transmission is the most energy intensive activity of wireless devices, maximizing alarm hit rate causes increasing energy consumption!

What can be done to achieve a trade off between those two goals?

Basically, reducing energy consumption of the restricted device can be achieved by reducing the workload:

- reduce data processing cost
- reduce data transmission/communication cost

Data processing cost can be reduced by screwing down the complexity of the classifier, this seems to be a valuable solution only in the first moment: **the simpler the classifier, the more false-alarms are generated**. This is founded in the pessimistic approach of the classifier, the algorithm has to raise an alarm – even if unsure: better raise a false alarm than ignoring or overlooking of a true alarm. More false alarms result in more unneeded transmissions, so using a too simple classifier is contra productive here.

Additionally, the classifier has to **adapt its behaviour** to the individual patient which is monitored, so even the classifier raises false alarms in the beginning, it has to learn and adapt it's future decisions to increase efficiency to a possible maximum. Fix installed rule sets are not applicable.

In fact it seems to be a good solution using the *most* intelligent classifier as possible: even this results in increased processing cost, the more intelligent classifier should reduce false alarms by his better classification performance. To accommodate the increased processing cost of more intelligent classifiers, I would suggest to introduce a **dynamic sensor sampling frequency**: i.e. in the “normal” case (when the system is idle), the sampling frequency is reduced to a minimal value – only of the classifier finds a degree of danger the sampling frequency is increased.

Recapitulating, we can formulate the following rules for the restricted device:

- use the most applicable and most performant intelligent classifier (to reduce false alarms as much as possible)
- which results in less unneeded transmissions of false alarms
- introduce dynamic sampling frequency of sensor data to minimize energy consumption in idle mode (which works better the better the classifier works)

1.3 Computational Characteristics of Classifiers

- investigate energy consumption, real-time processing, incremental learning, etc. characteristics of learning algorithms
- develop / find a technique, tool or method to measure computational characteristics of algorithms in a hardware independent way

1.4 Increasing Energy Efficiency of Classifiers

1.4.1 Existing/Related Research

There is ongoing research within the field of mobile/pervasive computing which aims on saving energy and/or balancing processing on mobile devices. The main targets of these efforts slightly differ from the requirements of a mobile monitoring system:

- existing research only focuses on energy saving, therefore processing intensive tasks are migrated to a remote server to save battery power – in our case this is not possible (todo: explain why; sequential data vs. “static” data, energy consumption of streaming sequential data)
- ...

1.4.2 Data Compression

Data Compression for Local Processing

Based on outcomes in [TSI00], data can be compressed without loss of effectiveness. This could be a key point for reducing processing costs on the mobile node (if compression is less costly than

analysis). This assumption has to be investigated. The main question is:

- **are the costs for local data compression less than the costs saved by compression?**

Data Compression for Remote Processing

If the monitoring device is in remote processing mode, all input data has to be transferred as quickly as possible to the backend device. For time-critical monitoring applications like health-monitoring it would not be feasible to wait until a useful amount of data for compression is collected, data has to be transferred as new values arrive.

The proposed solution to achieve both goals – send gathered input data in real time, perform compressed transmission of this data – I would suggest a simple algorithm (somehow related to “Run Length Encoding”, RLE, -- reference???):

- for every series of input values (a series represents a single sensor):
- transmit all contents of input data pool when switching to remote node (empty all buffers)
- after that, only transfer changed values for input series – if the monitored series doesn't change, don't transmit anything
- to allow the receiving machine to check whether the connection is up and functional, send a “heartbeat” event every X time intervals.

This method allows minimal transmission sizes by allowing real-time processing also on the server machine. It is clear that this approach only works when the monitored data has no “random jumps”, but this should be the case when monitoring a persons health for instance (pulse doesn't jump from 70 to 10 to 25 to 90 to 20).

It has to be checked how the integration of a change-threshold (i.e. ignore minor changes of input values) affects the overall performance of the system.

1.4.3 Data Abstraction

Russ suggests the application of data abstraction methods to reduce computational costs for analysing/monitoring time-series data: transform DATA to STATE and STATE to STATUS [RUS95].

Distributed Learning Investigation

1.4.4 Dynamic Sampling

As the remote health monitor just has to perform checks if everything is “normal” in regular intervals, the sensor sampling frequency should be reduced in idle mode/normal state to minimize power consumption. As the classifier of the monitoring device detects an abnormal situation, the sampling frequency has to be increased to get more detailed information about the state of the monitored object/patient. In the ideal case the sampling frequency is increased depending on the degree of “danger” the classifier identifies.

Data compression and data abstraction can be defined and programmed by the system architect / the programmer independently of the classifier used, dynamic sampling frequency models also depend on the abilities of the classifiers. For instance this is not really applicable for boolean classifiers which would turn sensor data sampling just on or off. The sampling frequency has to be adaptive to the degree of alarm the classifier identifies.

1.4.5 Energy Scavenging

Recent research brings up a very interesting way of how to tackle the problem of limited energy on mobile/embedded monitoring systems: energy is “scavenged” from the environment of the monitoring device. Lo and Yang mention recent work on vibration, temperature difference, electromagnetic field, light and infra-red radiation based energy generation ([LO05a]).

1.5 Backend Node

- has historical data of the patient
- has (anonymous) data of other patients (medical DB)
- has clinical personnel as human experts available

1.6 Local vs. Remote Processing

“Delegation of Control” or the problem of local versus remote processing depends on energy- and application related conditions (battery sparing, processor capacity, reaction time issues, etc.). A system has to be developed to decide *where* the analysis of data should take place and why. Martin et al derived a general measure for local vs. remote processing decisions ([MAR03]), the analysis

Distributed Learning Investigation

focuses on power-aware computing exclusively. Issues like maximum reaction time of the application, etc. are missing in their investigation:

1.6.1 Related Research

There are ongoing research efforts on energy aware mobile/pervasive computing issues. Some of the researchers are also investigating the question of saving energy due to migrating tasks to a powerful remote device. They mainly try to build hardware and application-independent frameworks to be utilized by mobile device software developers (e.g. software for laptops, PDAs).

These frameworks build their decision making process (local or remote processing) mainly on monitoring of execution characteristics, analysis of that data and building up new/adapted rules for future executions (see [FLI02], [RON03], [RUD98a], [RUD98b]). This run-time overhead is caused by the ignorance of the application that will make usage of the framework.

As the resources are very limited on a remote monitoring device and there is no need to install 3rd party applications, I would suggest another approach in our case: as the monitoring algorithm is chosen at design stage of the product and there are no dynamic software updates on the monitor node after being delivered, a pre-production analysis of execution characteristics can be performed.

For sure the result of this analysis is hardware and application dependent, but neither the hardware nor the software are target of major changes after the product is introduced on market (i.e. product design changes require new analysis of execution characteristics). Based on this data the local vs. remote processing decision making parameters can be tuned and fix installed at the production-series of remote monitoring devices.

This results in reduced efforts (processing costs, memory, battery) for the monitor device in production use.

Basic equations of Martin et. al.

Considering the question of local vs. remote processing only from the energy related view, a rule can be derived very easily. Basically, the energy consumed on the restricted (local) device should be minimized. We have to calculate energy required for local (E_{local}) and remote (E_{remote}) processing, the minimal value tells where the processing should take place:

Martin et al based their findings on a very simple basic expression:

Distributed Learning Investigation

$$E_{local} = (P_{systemon} + P_{cardsleep}) * T_{system} \quad E_{remote} = (P_{systemon} + P_{cardon}) * T_{trans}$$

where $P_{systemon}$ is the power of the device when processing, $P_{cardsleep}$ is the power of the device when communication is in sleep mode (not transmitting), T_{system} is the time required to complete the processing locally and P_{cardon} is the power of the device when communication is active (transmitting, both directions: sending and receiving), T_{trans} is the time required to complete the job remotely.

Clearly, above expressions are neglecting some of the circumstances of the distributed environment, such as network latency, a faster remote device (different local/remote processing times), network conditions, etc., etc. In a more detailed investigation these issues were integrated into the expressions (I will skip to write down each of the single steps here), as a result Martin et al found out that the figure computation_time / dataset_size ratio is a good indicator for local or remote processing.

A general derived rule says that candidates for remote processing have small transmission size and long computation time, candidates for local processing have long transmission size and short computation time. ([MAR03])

1.6.2 Energy related issues

Used the basic equations of [MAR03] as a starting point, we can develop our mathematical model (which has a different focus than related research as described above):

$$E_{local} = P_{local} * T_{system}$$

$$E_{remote} = E_{send} + E_{wait} + E_{receive}$$

As sensor data for processing has to be made available in both cases, the energy consumption for gathering sensor data is neglected.

When the algorithm on the monitor node decides to migrate processing to a remote backend server, the energy consumption equation stated above is not exactly true: to be accurate, there is a constant missing which tells the energy required to adapt the AI after results from the backend node are received (i.e. if the monitor raised a false alarm, it's AI algorithm has to perform a learning step – sometimes this can require lots of processing). More exactly, the equation for remote processing

Distributed Learning Investigation

energy consumption would be:

$$E_{remoteComplete} = E_{send} + E_{wait} + E_{receive} + E_{processResults}$$

But as $E_{processResults}$ strongly depends on the AI used and is expected to have an overall small fraction of the complete energy consumption, I will neglect it for now.

Power consumption of local processing has to be split up into:

$$P_{local} = P_{procActive} + P_{commIdle}$$

Further, we can assume that the energy required for sending data to a remote server and receiving the results, which are:

$$E_{send} = P_{send} * T_{trans} \quad \text{and}$$

$$E_{receive} = P_{receive} * T_{recv}$$

require the same (average) power consumption for tx/rx. We just continue with one figure for communication related energy: $E_{communication} = P_{commActive} * T_{trans}$ whereas T_{trans} is $T_{trans} + T_{receive}$.

In difference to related research we do not migrate a well-isolated processing task to the remote server (i.e. there is no static data like a file, etc which has to be synchronized) – in our case if the system decides to perform remote analysis of the sensor input, data has to be streamed all the time while the connection is open. I.e. we have no clear send -> process -> receive flow. In contrary to related research we cannot process the energy required to complete the task (as other projects are investigating well-defined tasks like compiling a program, formatting a latex document, etc) – we only can focus on the energy required to process sensor input within a duration of time (there is no actual end of the task “monitor the patient”).

Our formulas transform to:

$$E_{local} = (P_{procActive} + P_{commIdle}) * T_{system} \quad \text{and}$$

$$E_{remote} = (P_{procIdle} + P_{commActive}) * T_{trans} + (P_{procIdle} + P_{commIdle}) * T_{wait}$$

Whereas T_{system} is the time required to process a input vector locally, T_{trans} is the time required for communication with the backend system (send and receive) and T_{wait} is the time the monitor device has to wait while the backend system processes the data. When $T_{trans} = S/r$ whereas S is the Size of data to be transmitted and r is the transfer rate, we can transform the equation for remote

Distributed Learning Investigation

processing to

$$E_{remote} = \frac{(P_{procIdle} + P_{commActive}) * S}{r} + (P_{procIdle} + P_{commIdle}) * T_{wait}$$

Investigations of current hardware (like the wavecomm wismo quik series) showed, that modern GSM/GPRS modems can be turned off when not used, so we can assume that power consumption of idle communication is zero:

$$P_{commIdle} := 0$$

so

$$E_{remote} = T_{wait} * P_{procIdle} + \frac{(P_{procIdle} + P_{commActive}) * S}{r}$$

Dynamic Sampling Frequency and Degree of Abnormality

The proposed system architecture for building an energy efficient, intelligent remote monitoring systems includes the technique of “dynamic sampling frequency”. Imagine the monitor node: its task is to watch an object of interest (e.g. the patient) and identify if there are exceptional circumstances or not. If not, everything is ok – which means: save energy, do nothing more. IF exceptional circumstances are identified (i.e. an emergency, alarm), the monitor has to react.

So, the events of interest are the emergencies, not the normal situations. The majority of the time the monitor is working, there will be NO emergency – we have to minimize energy consumption within these periods of time. Dynamic sampling addresses this issue:

If there is a normal situation, it is not needed to read and process sensor data very often (“often” is depending on the application). If we are talking about monitoring a persons health, I would assume that in the “normal state”, reading and processing sensor data every minute would be enough. In contrary to an emergency, where we need to analyse detailed information about the characteristics and gradient of the input data, we may require to read sensor data multiple times a second.

What we need is a technique to adjust this dynamic sampling frequency (F_s) in relation to the estimation of the classifier, I'll entitle it simply

“degree of abnormality”: D

As described above, we are not interested in anything is everything is normal. The farer the

Distributed Learning Investigation

situation gets from normal, the more detailed information about the environment is needed. So if everything is absolutely normal ($D=0$), sampling frequency and local processing is reduced to an absolute minimum. On the other hand, if we have an absolutely abnormal (=emergency) situation ($D=1$), we need maximum information from the inputs, send alarm data to a backend system and do not really care about energy consumption. With D we have a tool to influence the dynamic sampling frequency and support the decision making process of the monitoring AI whether it should process sensor data local or remotely:

- depending on the value of D , we increase F_s
- if D exceeds a threshold value D_t , the monitor opens up the communication line to the backend system and performs remote processing of input data

Introducing the threshold value for remote processing has two advantages: first, we can define a degree of emergency (which is D) where the monitored patient is in a state of danger at which we definitely want to have an alarm raised, i.e. we have no boolean alarm logic (alarm / no alarm), we can achieve a “fuzzy” alarm decision by utilizing D as an indicator for the alarm.

The second advantage is that the remote node starts receiving sensor input at an earlier stage of the alarm (or even before a true alarm is upcoming). This allows a better analysis of the development of sensor data over a longer period of time on the more powerful (and more intelligent) remote node.

How the sampling frequency is influenced by D depends on the application design and the environment that is being monitored, an example could be:

$$F_s = F_b * \left(2 - \frac{1}{(1+D)^2}\right)$$

Where F_b is the “base frequency”, used in normal state: $D=0 \rightarrow F_s = F_b * 1 = F_b$. Using the equation would increase the frequency up to 1,75 times of F_b .

For sure there are more adequate approaches than the one above but this is not the main point of interest now.

Integration of Data Abstraction and Feature Selection

We already identified two programmatic approaches to reduce local processing efforts: data abstraction and feature selection (as mentioned in [DIT02], [TSI00]):

Distributed Learning Investigation

Data Abstraction

Data abstraction means to build symbolic input values rather than concrete sensor measurements, ie build ranges and assign them to the inputs, for example:

- no_pulse = 0-10
- slow_pulse = 11-60
- normal_pulse = 61 – 90
- high_pulse = 91 – 110
- very_high_pulse = gt. 110

It is clear that the above categorization of input data extremely reduces complexity for the classifier, the combinations of different input values are extremely reduced.

Feature Selection

Feature selection means to calculate a value representing a time-series of input values rather than using every single measured input value, for example calculate the average, the median or the mean of a series of input values and use the result as input for the classifier.

Data Abstraction and Feature Selection are methods to reduce processing cost on the restricted mobile monitor device. Clearly, they also blur the input in a certain way. To allow exact analysis of the sensor data in the case of an alarm, feature selection and data abstraction should not be used when data is streamed to the remote backend node. It should be incumbent on the remote algorithm itself how the data is pre-processed, the monitor node should transfer the measurements 1:1. Therefore **feature selection and data abstraction only affect the energy requirements for local processing.**

$$E_{local} = E_{prepare} + E_{process}$$

where prepare means feature selection and data abstraction and process means the classification algorithm (the AI).

$$E_{local} = T_{prepare} * P_{procActive} + T_{process} * P_{procActive} = (T_{prepare} + T_{process}) * P_{procActive}$$

Now we have to take the sampling frequency into account. Increased sampling frequency produces a greater series of input values for a period of time, as the classifier on the monitoring node already gets a single input value representing a time-series (the “feature”), increasing the sampling

Distributed Learning Investigation

frequency increases the efforts for data abstraction and feature selection:

$$T_{prepare} = T_{abstraction} + T_{featureSelection} = F_s * c * T_{prepare}$$

Regardless of which technique is more influenced by the sampling frequency (it makes a difference if abstraction is performed before feature selection or the other way round), I introduced the influencing coefficient “c” which is a constant value that describes the way how data abstraction and feature selection is influenced by the sampling frequency.

$$E_{local} = (F_s * c * T_{prepare} + T_{process}) * P_{procActive}$$

Representing Processing Time

All of above equations strongly depend on time-based values like transfer-time and processing time. The required time to transfer data is already expressed by the throughput, data size divided by bandwidth rate r . As I don't want to rely on a specific hardware platform at this stage of analysis and processing times can only be exact values when being measured on concrete hardware, I'll express the time required for processing as follows:

$$T_{process} = \frac{\text{complexity/size of the algorithm}}{\text{performance figure of the hardware}} = \frac{\sigma}{v_p}$$

Where σ is an abstract value representing the complexity of the algorithm (e.g. number of processor instructions, etc) and v_p the performance figure of the hardware platform (e.g. MIPS rate, etc).

Including this representation into our equations for local and remote processing efforts we get:

$$E_{remote} = T_{wait} * P_{procIdle} + \frac{(P_{procIdle} + P_{commActive}) * S}{r}$$

$$T_{wait} = \text{remote processing time} = \frac{\sigma_{remote}}{v_{pRemote}}$$

$$\rightarrow E_{remote} = \frac{(\sigma_{remote} * P_{procIdle})}{v_{pRemote}} + \frac{(P_{procIdle} + P_{commActive}) * S}{r}$$

and

$$E_{local} = \frac{(F_s * c * \sigma_{prepare} + \sigma_{process})}{v_{pLocal}} * P_{procActive}$$

Integration of data compression when remote processing

We must not forget to integrate cost of data compression for remote processing equations:

$$E_{compress} = T_{compress} * P_{procActive} = \frac{\sigma_{compress} * P_{procActive}}{v_{pLocal}}$$

As, similar to data abstraction and feature selection in local processing mode, data compression depends on the amount of input data, we have to take the dynamic sampling frequency into account:

$$E_{compress} = \frac{F_s * d * \sigma_{compress} * P_{procActive}}{v_{pLocal}}$$

$$\rightarrow E_{remote} = E_{compress} + \frac{(\sigma_{remote} * P_{procIdle})}{v_{pRemote}} + \frac{(P_{procIdle} + P_{commActive}) * S}{r}$$

$$E_{remote} = \frac{F_s * d * \sigma_{compress} * P_{procActive}}{v_{pLocal}} + \frac{(\sigma_{remote} * P_{procIdle})}{v_{pRemote}} + \frac{(P_{procIdle} + P_{commActive}) * S}{r}$$

Application of data compression somehow reduces the amount of data to be transferred (S), the compression rate:

$$compression\ rate = \frac{S_{compressed}}{S}$$

As the compression rate itself depends on the data being compressed, this is an unknown variable: s (from “shrink”):

$$E_{remote} = \frac{F_s * d * \sigma_{compress} * P_{procActive}}{v_{pLocal}} + \frac{(\sigma_{remote} * P_{procIdle})}{v_{pRemote}} + \frac{(P_{procIdle} + P_{commActive}) * S * S}{r}$$

Use a slow or a fast processor ?

The equations modelled above provide an easy way to decide the better alternative:

- use a slow but power-saving processor
- or use a power-hungry but very fast processor

on the restricted mobile device. The fast but power-hungry processor consumes less energy than the slow processor if:

$$E_{fast} < E_{slow} = \frac{P_{fastProc} * (F_s * c * \sigma_{prepare} + \sigma_{process})}{v_{fastProc}} < \frac{P_{slowProc} * (F_s * c * \sigma_{prepare} + \sigma_{process})}{v_{slowProc}}$$

$$\rightarrow P_{fastProc} * (F_s * c * \sigma_{prepare} + \sigma_{process}) * v_{slowProc} < P_{slowProc} * (F_s * c * \sigma_{prepare} + \sigma_{process}) * v_{fastProc}$$

$$\rightarrow P_{fastProc} * v_{slowProc} < P_{slowProc} * v_{fastProc} = \frac{P_{fastProc} * v_{slowProc}}{P_{slowProc}} < v_{fastProc}$$

Then we can define the speed of the fast platform in times of speed of the slow platform:

$$v_{fastProc} = x * v_{slowProc}$$

which results in

$$\frac{P_{fastProc} * v_{slowProc}}{P_{slowProc}} < x * v_{slowProc}$$

$$\rightarrow x > \frac{P_{fastProc}}{P_{slowProc}}$$

This can be interpreted as: **the speed of the fast processor has to be greater than the proportion of its power consumption divided through the power consumption of the slow processor to perform better with less energy consumption.**

An Example:

- Slow processor: average power cons of 50mA, performance figure of 5
- Medium processor: average power cons of 85mA, performance figure is 8
- Fast processor: average power cons of 120mA, performance figure is 17

Comparing the slow and the medium processor:

$$x = 8/5 = 1,6$$

power consumption ratio is $85/50 = 1,7$

$$x = 1,6 > \frac{85}{50} = 1,7 \rightarrow 1,6 > 1,7$$

which is false, 1,6 is not greater than 1,7 --> in this case the slow processor would be more energy efficient.

Comparing the slow and the fast processor:

Distributed Learning Investigation

$$x = \frac{17}{5} > \frac{120}{50} = 3,4 > 2,4$$

which is true. In this case, the fast processor would be more energy efficient even if its power consumption is more than 2 times greater than the slow processor.

1.6.3 Further Issues

Considering only energy relevant issues will maximize battery runtime but won't result in a usable application. Certainly other issues as a maximum reaction time (T_{max}) of the application, etc. have to be considered. Thinking of a remote monitoring application and the major aim stated above (p. 8), what are the reasons why the remote device should delegate control to the backend node?

- if energy required for local processing is bigger than energy required for remote processing:

$$E_{local} > E_{remote}$$

- if time required for local processing is greater than allowed maximum processing time and remote processing time (time for transmission and waiting) is less than local processing time: $(T_{local} > T_{max}) \wedge (T_{remote} + T_{trans} < T_{local})$

- if an alarm is identified: $C = alarm$
- if classifier is “unsure”: $C = unknown$
- if the “degree of abnormality” is greater than a defined threshold value (the degree of abnormality somehow models the “unsureness” of the classifier): $D > D_t$

1.7 Knowledge Exchange

The restricted resources enforce simple learning algorithms on the restricted node, but on the back end or assistance node we have completely different conditions: processing power, energy, storage capacities for masses of data, clinical experts. Therefore it is feasible to build complex learning and data mining algorithms which perform better than the mobile learning ones. Building a back end/assistance node with the same learning algorithms as the mobile node would not make sense at all.

This design comes up with two different learning algorithms: a simple algorithm on the mobile node, a more sophisticated one on the back end node. Learning algorithms and machine learning models have their own representation of the knowledge gained, how can two different learning algorithms communicate their gathered knowledge? Regardless of the implementation, the

Distributed Learning Investigation

knowledge of the mobile node (K_m) has to be a subset or equal to the knowledge of the assistance node (K_a ; the assistance node can know “more” due to its enhanced resources):

$$K_m \subseteq K_a$$

There are mainly two approaches of how to transfer knowledge between both nodes:

1. both algorithms are based on a common representation of knowledge
2. one node is “client” of another node, i.e. the “more intelligent” node trains the other.

1.7.1 Knowledge Representation of Classifiers

Investigate how classifiers represent/store their knowledge --> this could result in base-knowledge for the knowledge-exchange problem.

1.7.2 Mutual Training of Classifiers

Backend node acts as trainer/supervisor for the mobile node and vice versa.

1.8 Communication / Data Transmission

If the remote monitor identifies an abnormal situation, the GSM/GPRS line is opened up and sensor data is streamed in real-time to the backend server to allow detailed analysis. It has to be considered if data compression of the streamed real time data is applicable – power consumption of the mobile device while streaming the data must be minimized, i.e. as long as the bandwidth capacities are enough no compression should be used to avoid additional energy consumption through calculating the compression algorithm (at least this also produces slight delays of the sent real time data).

Maximum transfer rates for remote devices represent basic conditions for the amount of data being processed at maximum. Maximum rates have to be investigated:

- GSM->GPRS
- GSM->EDGE
- UMTS
- upcoming new technologies/standards (4G)?

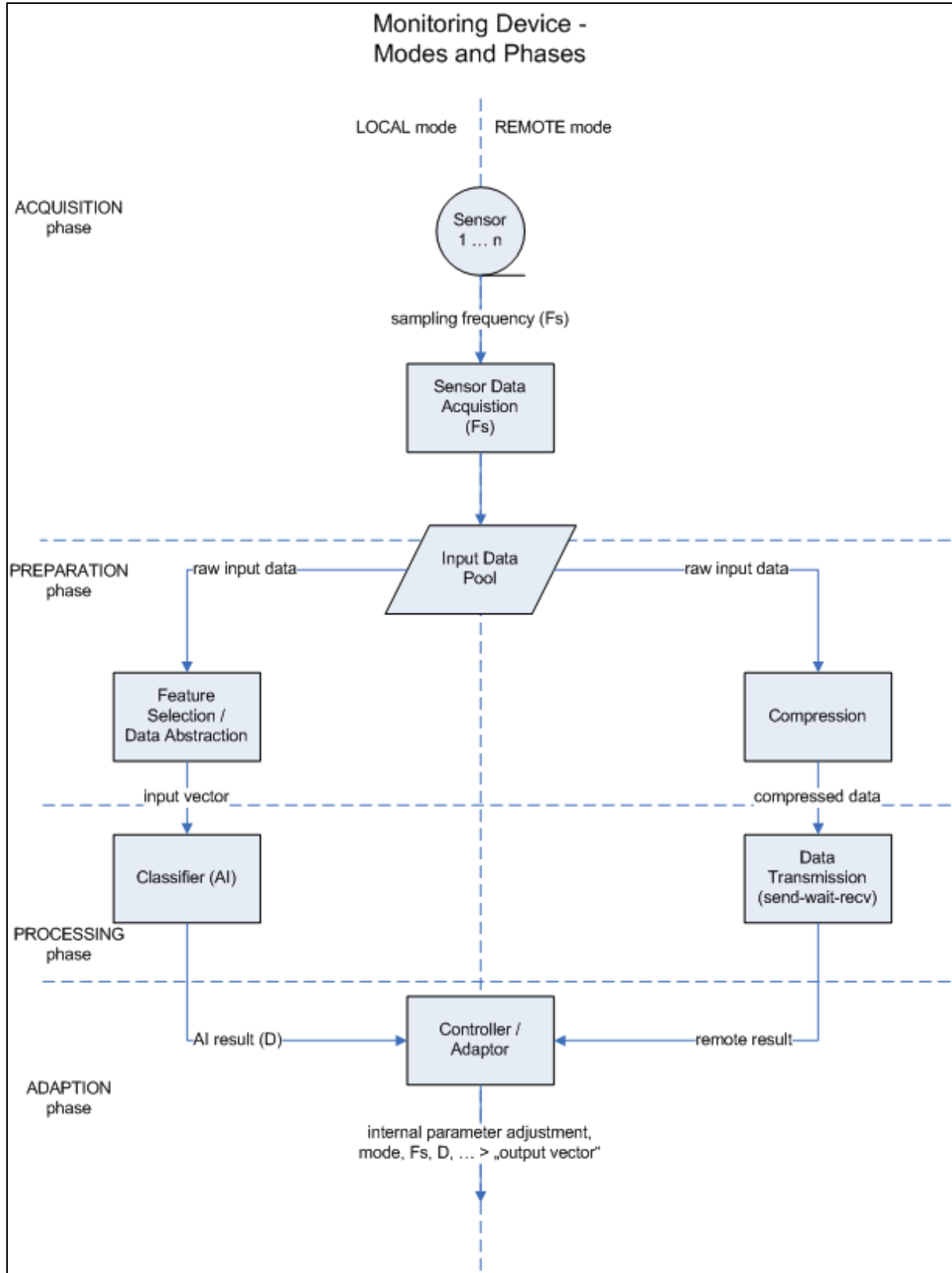
Distributed Learning Investigation

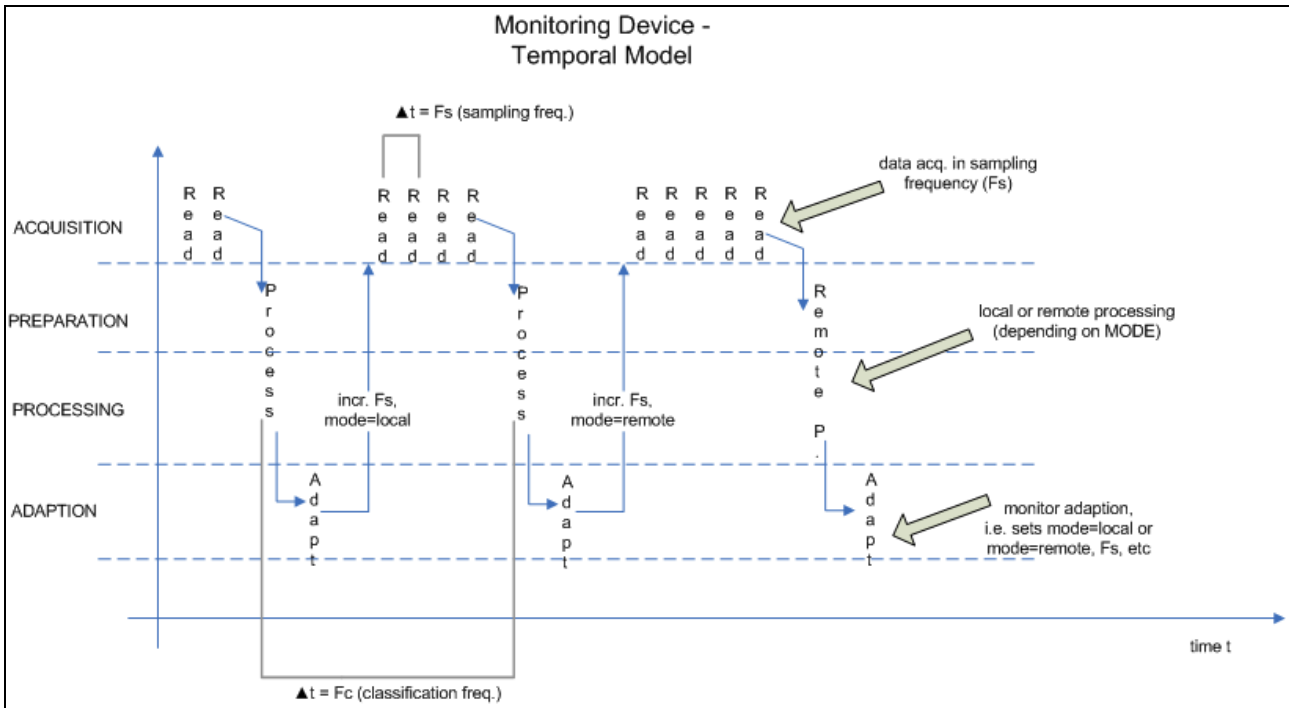


- nearer description of data transmission related issues and research targets
- e.g. how many sensor data is available – can all be transferred to the backend?

2 Appendix

2.1 Schemes of the proposed monitor model





2.2 References

- [MIT97] Mitchell, Tom M.; Machine Learning, McGraw-Hill, 1997
- [TSI00] Tsien, C.L. TrendFinder: Automated Detection of Alarmable Trends 2000
- [RUS95] Russ, T.A.; Use of data abstraction methods to simplify monitoring, , 1995
- [LO05a] Lo, Benny; Yang, Guang-Zhong; Key Technical Challenges and Current Implementations of Body Sensor Networks, , 2005
- [MAR03] Martin, T.; Siewiorek, D.; Smailagic, A.; Bosworth, M.; Ettus, M.; Warren, J.; A Case Study of a System-Level Approach to Power-Aware Computing, ACM, 2003
- [FLI02] Flinn, J.; Park, S.Y.; Satyanarayanan, M.; Balancing Performance, Energy, and Quality in Pervasive Computing; Proc. of the 22nd Int. Conf. on Dist. Comp. Systems; IEEE Society, 2002
- [RON03] Rong, P.; Pedram, M.; Extending Lifetime of a network of Battery-Powered Mobile Devices by Remote Processing (...); Proc. of the 40th conference on Design automation; ACM, 2003
- [RUD98a] Rudenko, A.; Reiher, P.; Popek, G.; Kuenning, G.; The Remote Processing Framework for Portable Computer Power Saving; Proc. of the '99 Symposium on Applied Computing; ACM, 1998
- [RUD98b] Rudenko, A.; Reiher, P.; Popek, G.; Kuenning, G.; Saving Portable Computer Battery Power Through Remote Process Execution, ACM, 1998
- [DIT02] Dietterich, T. G.; Machine Learning for Sequential Data: A Review, Caelli, T., 2002