14

Control Theoretic Approach to Platform Optimization using HMM

Rahul Khanna[1], Huaping Liu[2] and Mariette Awad[3]
[1]*Intel Corporation, 2111 NE 25th Ave., Hillsboro, OR 97124*
[2]*School of EECS, Oregon State University, Corvallis, OR 97331*
[3]*American University of Beirut, (ECE) Riad El-Solh, Beirut 1107 2020*
[1,2]*USA*
[3]*Lebanon*

1. Introduction

Power optimization and power control are challenging issues for server computer systems. A system can be represented as a set of components whose cooperative interaction produces useful work. These components may be heterogeneous in nature and may vary in the power consumption and power control mechanisms. Server system components may coordinate power control actions using embedded controllers or special hardware. System development tends to be a complex process that competes for performance in the presence of design constraints. These constraints may be on manufacturing cost, validation cost, area, form-factor, or operational costs. Operational cost is related to the cost of operating a system for a unit of work. Operational cost reduction requires observability, controllability, and adaptability. These features come at a price that may increase the manufacturing, design, and validation costs.

Energy efficient design helps in realizing a system that minimizes power and thermal dissipation for a given performance constraints. These systems can perform one or many functions related to power/thermal management for a given performance policy: 1. Parameter tuning to reduce energy consumption for a given performance policy. This may require collective (or coordinated) tuning of system components for minimum power usage at given performance levels. 2. Limiting the power of an individual component (or set of components) in a power constrained system. Power is allocated (or de-allocated) in a manner such that performance degradation is minimized to the extent possible. 3. Power prediction and forecasting to avoid sudden state changes. This prediction can be at the component level or at the system level. For example, we may predict the inactivity periods between bursts of memory traffic, which allows us to proactively prepare the system for an appropriate sleep state. This avoids reactive latencies and hence increases performance. 4. Distributing the available power to system components in a manner that maximizes the overall performance. One strategy may involve individual allocation (or de-allocation) due to each component's share in performance gain. 5. Using activity vectors to perform thermally balanced computing, thus avoiding hot spots. Activity data can also be used to co-schedule tasks in a contention-free and energy-efficient manner.

Furthermore, energy-efficient systems design involves complex choices due to a variety of degrees of freedom for power parameter tuning. The process involves modeling methodology, implementation choices, and dynamic tuning. Modeling methodology includes the choice of algorithms or heuristics that tunes the state transition. Implementation choices involve the hosting of executable code in a manner such that it can access the appropriate telemetry data in an efficient manner at runtime. Additionally, it should have enough computation power to perform policy-related functions while being non-intrusive during sleep states.
In recent years energy-efficient design in servers has received much primarily due

- The need to reduce heat dissipation, thereby reducing the cooling costs
- The need to reduce energy consumption, thereby reducing the energy-related operating costs
- Strict current limits in a power-limited server rack. It may therefore be desired to maximize the rack consumption while keeping the energy limits within regulations
- Capacity planning that requires efficient use of existing real-estate, which necessitates the optimal use of available racks.

In general, energy efficient design helps in realizing a system that minimizes power and thermal dissipation for a given performance constraints. These systems can perform one or many functions related to power/thermal management for a given performance policy:

- Parameter tuning to reduce energy consumption for a given performance policy. This may require collective (or coordinated) tuning of system components for minimum power usage at given performance levels.
- Limiting the power of an individual component (or set of components) in a power constrained system. Power is allocated (or de-allocated) in a manner such that performance degradation is minimized to the extent possible.
- Power prediction and forecasting to avoid sudden state changes. This prediction can be at the component level or at the system level. For example, we may predict the inactivity periods between bursts of memory traffic, which allows us to proactively prepare the system for an appropriate sleep state. This avoids reactive latencies and hence increases performance.
- Distributing the available power to system components in a manner that maximizes the overall performance. One strategy may involve individual allocation (or de-allocation) due to each component's share in performance gain.
- Using activity vectors to perform thermally balanced computing, thus avoiding hot spots. Activity data can also be used to co-schedule tasks in a contention-free and energy-efficient manner.
- Profiling task characteristics related to (a) Task priority (b) Energy and Thermal profile (c) Optimization methodology regarding latency targets proportional to task priority.

2. HMM approach

We face several challenges in the establishment of power/thermal monitoring infrastructure that can uncover complex deviance from an established norm. The correlation of sensors is typically separated by a significant amount of time that makes it difficult to model. In such cases, the Hidden Markov Model (HMM) is particularly useful because it can exploit

the pattern in the sequence of events to predict the state. Since HMM uses and correlates observations with hidden states, it may very well be a consideration in system design. Observation points can be optimized using a reasonable set of system-wide QoS checkpoints (QC) or sensors. Hidden states can be created using explicit knowledge of probabilistic relationships with these observations. These probabilistic relationships, which are also called profiles, are hardened and evolved with the constant usage of the multiple and autonomous systems. These profiles, HMM parameters and observation probability density function (pdf) are stored in a central storage where they are re-estimated based on the HIT/MISS data collected. If observation checkpoints can be standardized, then the problem of predictability can be reduced to profiling the existing and new hidden states to standard observations. In terms of modeling a large number of temporal sequences, HMM can serve as an excellent alternative, because it has been widely used for speech recognition, image identification, microbiology, Internet attacks, and misuse based on operating system calls. In all these applications, the aim is to map a pattern to one of the many states. If we consider an abnormal behavior to be a pattern of an observed sequence, HMM should be appropriate to map those patterns to one of several states. Furthermore, it is essential to build an adaptive strategy based on embedding numerous policies that are informed by contextual and environmental inputs. The policies govern various attributes of behavior, enhancing flexibility in order to maximize the efficiency and performance in the presence of high levels of environmental variability.

Power autonomic environment comprises of cooperating elements that play an optimization game among themselves in order to optimize its resource usage while adhering to the global policies related to power and thermal constraints. A real-time mechanism can to be devised independent of the behavior of the cooperating components. This mechanism enhances the ability to detect abnormalities or policy drifts by monitoring unusual activities (or drifts) in the system by comparing it to a user's profile. It analyzes the trending in a system's behavior as it evolves with usage. We may utilize relevant observations (such as changes in system performance parameters or fault frequency) to predict hidden states (operational degradation, performance loss). These methods detect and report system abnormality as a result of drift for an acceptable profile. According to Denning (Denning, 1987), a profile characterizes the behavior of a given subject with respect to a given object, thereby serving as a signature or description of normal activity for its respective subject(s) and object(s). Observed behavior can be characterized using statistical metric represented by a random variable x monitored over a period of time. Metric can typically be defined using a counter, interval and evaluation data. Observations records along with statistical models can be used to determine the conformity of the normal process (or activity). Two of the profiles that aid in QoS analysis are the following:

- System Usage Profile. This profile is the summary of system usage trends for a given subject (or user DNA). It contains information related to users' best-known practices (or trends) such as most used application, average system activity, or user preferences.
- System Activity Profile. This profile is the summary of system resource usage trends over a period of time (or system DNA). It contains information related to resource usage (CPU usage, memory usage, input/output (IO) usage, number of tasks, execution-time, and system calls).

Any deviation from a standard profile identifies a potential anomalous state and triggers a policy based method supported by autonomics. A false positive trigger acts as a feedback for retraining the model. A functional approach would correlate the system observations (using usage and activity profile) and state transitions to predict the most probable state.

The state in this case describes the component's ability to operate constructively in an optimization game for a given objective. These objectives could be related to improving system's performance/watt for a given power limits, reducing performance de-gradation as a result of component fault, or identifying the components that have been compromised as a result of intrusions or other events.

2.1 Power optimization in server platforms

This chapter presents a framework of modeling and optimization for stochastic power management using HMM. It describes the essential ingredients that is responsible for building the profile and detecting any deviations from that profile. The objective is to anticipate the power/thermal policy deviations while reducing the number of false positives. The model has to fulfill several objectives related to accurate profile deviation detection using different ingredients such as:

- QoS checkpoints (QC) that analyzes the sensor activity which predicts the transition from a normal state to an abnormal state.
- Creation of an activity profile that identifies the abnormal activity of the observable states by measuring the sensor deviation from the normal behavior. In short, it characterizes the behavioral signature corresponding to the normal activity of a given subject with respect to a given object.
- Concept drift that measures the change in the user behavior over a period of time.
- Control loop that adapts the checkpoint trigger according to the weighted sum of proportional, average, and derivative sensor measurements over derivative and integral time window.
- Model that predicts the most probable state based on previous state (normal/abnormal) as well as observed states.

HMM based model utilizes the distributed checkpoints in a platform. These checkpoints are in a form of sensors, activity counters, error counters, performance counters and energy counters. A series of observations from these checkpoints are utilized to identify the HMM model applicable for an evaluation period that describe the workload behavior. Workload behavior is a characteristics of the resource utilization patterns. For example, workload may be I/O bound, CPU bound, Network bound or idle. Workload behavior patterns are then used to tune the system for optimal performance/watt. The methodology essentially use the HMM based stochastic power management techniques to tune the system effectively in order to:

- Avoid reactive response to power state change demand.
- Optimal distribution of power states between silicon components according to the performance gains.
- Optimal tuning of the entry/exit timings of the power states according to the workload profiles.
- Optimal operation of the fans by evaluating the effectiveness of the changes in Fan speeds wrt. component heating, power consumption and performance variations.

Depending upon the complexity of workload relationship to the utilization of several active components in a system, the solution space could involve large number of variables with high dimensionality. These variables tend to be noisy and discontinuous with little or no

information about the corresponding interactions between there respective components. The problem is further aggravated by the fact that hardware/software based power/thermal optimization functions kicks in autonomously, thus making the complete solution highly non-linear. The effectiveness of a potentially significant variable for one workload may cease to be effective for other depending upon the underlying relationship to the output (estimated power). Conversely, formerly insignificant variable may play a significant role in determination of estimated power in the newer workloads. Hence, before calibrating a non-linear equation with a set of selected variables, we also need to select an optimal set of variables within a large search space that will eventually lead to an accurate solution. Higher accuracy targets present a larger search space with a potential to uncover currently unknown non-linear behaviors.

2.2 Variable reduction & consolidation

Complex variables are essential ingredients of building the HMM model. They extract the observed states functions that are modelled to predict one of the HMM states (QoS, QDP or QV) (Figure 3). Variable Reduction (Fig. 1) helps to selects the weight for the input variables of a component (CPU, Memory, HDD etc.) after identifying those which: (1) Are most correlated with the output values, (2) Cause discontinuities and contribute to the threshold effects in the output values and (3) Are eliminated as they possess a high degree of linear correlation with another variable such that both influence the analyzed output variable in a very similar (if not an identical) way.

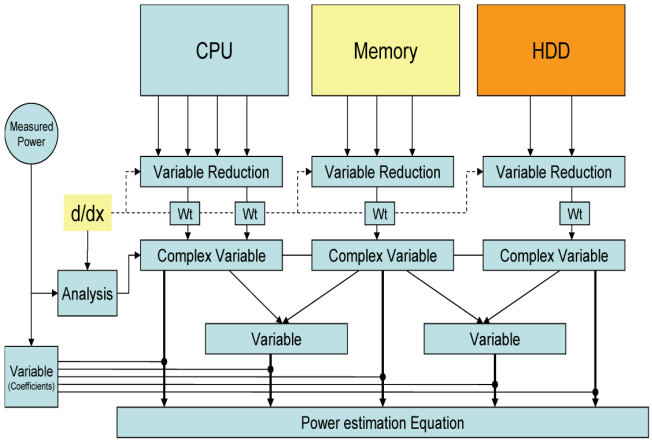


Fig. 1. Synthesis of optimal power estimation equation using variable reduction. Reduced variables construct complex variables that can then be used as emissions in the *Hidden Markov Model*

The main objective of variable reduction is to help analyze and compare the slope coefficients of a system's model where component's input variable is regarded as the most significant which is related to the slope coefficient of the largest absolute value. A large slope coefficient indicates high sensitivity to very small changes in the input variable that results in very large changes in the output variable. In our approach we extract consequetive models that are created by shifting sampling window of the data points. Each model shares the sampled data

with consequetive models. This helps in converting a multidimension non-linear model into a series of linear models:

$$\chi_i = \frac{dP_i}{dt} = \sum_{j=0}^{N}(\lambda_i^j * \frac{dV^j}{dt}), \quad \sum_{j=0}^{N}\lambda_i^j = 1 \tag{1}$$
$$t_i \in (i*d, i*d+M)$$

where, χ_i represents the estimated rate of change of power, P_i represents the estimated power of the i^{th} model, λ_i^j represents the coefficient of the j^{th} variable of i^{th} model, V^j represents the j^{th} input variable, and t_i represents the modeling window of the dataset of size M shifted by d samples. Coefficients λ_i^j of each model i is determined by curve-fitting using Genetic Algorithm (GA) with the fitness represented by:

$$F_i = \frac{1}{M}\sum_{t_i}|max(0, \frac{\hat{\chi}_i - \chi_i}{\hat{\chi}_i})| \tag{2}$$

where $\hat{\chi}_i$ represents the measured rate of change of power. λ_i^j represents the significance of the corresponding variable X^j for model i. Finally average of λ_i^j determines the overall significance of each variable as given by equation 3:

$$\lambda^j = \frac{\sum_{i=0}^{M}\lambda_i^j}{M} \tag{3}$$

Variable X^j of each component are selected based on the sensitivity of the variable represented by λ^j (Equation 3). Upon the completion of the sensitivity analysis, a reduced equation is formed for each component (CPU, memory, HDD) using the adjusted weighted sum $\hat{\lambda^k}$ of dominant variables V_x^k, where k represents the set of all selected dominant variables:

$$V_x = \sum_k \hat{\lambda^k} * V_x^k \tag{4}$$

Reduced set of variables form a distributed observables for identifying a change in a workload conditions that would require a control action to balance the power, thermal and performance parameters.

3. Attributes of adaptation

Adaptation is primarily achieved by steering each platform component to its optimal state. Normally, users are subjected to arbitrary performance and environmental variations. Adaptive systems exist to solve such problems that result due to a great deal of variability, flexibility, and dynamism. Adaptation may also serve functions that may be mutually hostile and pull in different directions. This results in making compromises between solutions in an effort to maximize the fitness of the overall solution. For example, increase in performance is generally associated with an increase in power that results in a further increase in power due to cooling pressures. An adaptation function will optimize the power in a manner that delivers the desired performance as perceived by the application. It is noteworthy that desired performance may not necessarily be the highest performance. In real systems, it is impossible to improve all aspects of the target policy to the same degree

simultaneously. Traditional adaptation techniques use single point optimizations and do not employ multiple adaptations synergistically. For complex platforms, adaptation approach migrates to multi-point space exploration that chooses the optimal configuration that can continue to achieve homeostasis. This requires a prediction-based classifier that determines the survivability of a particular configuration. An important step in adaptation is to generate on-line (or off-line) behavioral profiles of different configurations under varying resource conditions. Necessary requirements for the adaptation process are:

- The ability to monitor resource conditions in a continuous mode using sampling rates according to observed data redundancy and the control periods.
- The ability to determine when adaptation should be performed.
- The ability to determine how the adaptation should be performed by modeling control behavior.
- The QoS profile that describes the real-time constraints and its resource requirements for a given autonomic element.
- Choice of available execution paths for a given autonomic element.

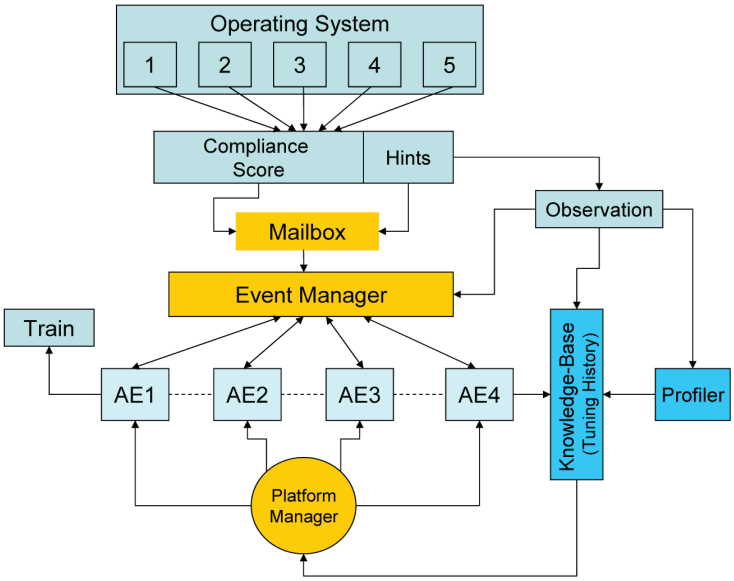


Fig. 2. Adaptation as a function of compliance scores/hints and the history of Adaptation. An autonomics manager acts as a conduit to the knowledge base. AEs make a process control decision using the previously tuned parameters based on applicability of the historical trends due to workload changes.

The QoS profile governs an appropriate level of resource reservation by indicating the output quality levels in a dynamic fashion. In general, the QoS maximization process starts with an initial resource allocation which it revises according to changing application demands and satisfaction levels. For example, applications can specify the satisfaction level at a scale of

0-9, with 9 being extremely satisfied. Additionally, applications can offer potential hints into the basis of performance loss. This information is consumed by the competing objectives functions that engage in an optimization game and generate an alternate configuration to maximize the application satisfaction score. Fig. 2 shows the adaptation infrastructure where an application produces a satisfaction score and associated hints.
Models may utilize these hints based on their applicability using an event manager filter. Once compliance score and hints are evaluated, each model identifies a set of tuning parameters that can maximize the compliance score. Resulting decisions steer each model to dynamically train itself based on the historical trends and scores. The dynamic training process transparently includes the competing strategy in order to play the optimization game that leads to a stable equilibrium. Traditional adaptation techniques (especially power optimization) have remained seemingly incognizant of the strategies employed by other components and the interplay between them. The adaptation techniques are built with an assumption that other components are not operational at the time of adaptation. Global optimization is only possible if tradeoffs between competing components are identified. In the absence of tradeoff study, only the most aggressive strategy, although suboptimal, may survive. We can characterize the operations of the various components as a non-cooperative dynamic game (Γ) played at interval T when game strategies are evaluated. Each player represents its strategy space by aggregating its goal management strategies. For example, the CPU can represent its power management strategy space using a set of N+1 "P" states, as illustrated in Equation 5:

$$S_{cpu}^{T} = [S_{cpu}^{0}, S_{cpu}^{1}, S_{cpu}^{2}, S_{cpu}^{3}, \cdots S_{cpu}^{N}] \tag{5}$$

Where S_{cpu}^{T} represents the strategy space that is not attached to any CPU power management scheme. The strategy space for other components is also represented in the same manner. Cumulative strategy space is used to optimize the usage of the shared resource as described by the global policy specification, as shown in Equations 6 and 7:

$$f(R(t)) \geq \sum_{i} (S_i^R(t)) \tag{6}$$

$$f(R(T+1)) = f(R(T)) - \sum_{i} (\delta_i^R) \tag{7}$$

Where, $S_i^j(t)$ represents the strategy at time "t" for component "i" and shared resource "j". $f(R(t))$ represents the policy specification of shared resource "R" at time "t". δ_i^R represents the reduction in resource "R" by component "i" in the subsequent interval (T+1). Consequently, higher δ_i^R may not directly translate to a higher quality. The objective is to determine a value for δ_i such that both the overall quality of service of the device is maximized.
Once automated, adaptation maximizes the performance of the target system without manual intervention. It tunes a system that trades off resource requirements over time for a desired level of quality of service. Furthermore, it provides flexibility in allocating resources to competing elements in a manner such that all objectives' goals meet their real-time requirements. Tuning History acts to correct the model throughout the life of the platform optimization objectives. Complex variables, compliance scores, hints and component performance counters act as standard emissions that construct the basis of HMM model. Furthermore, adaptation step is performed by changing the operating functions of cooperating component (power state, frequency, off-lining, changing idle timings etc.)

3.1 Control modeling ingredients

This section describes the essential ingredients for a profile detection system (PDS) that is responsible for building the profile and detecting any deviations from that profile. The objective of the PDS is to anticipate the power, thermal or performance policy deviations while reducing the number of false positives. An instantaneous deviation from a normal profile can be expected due to a momentary change in the system environment. Therefore in PDS we have to fulfill objectives related to accurate profile deviation detection using different ingredients such as:

- *Checkpoints* to analyze the sensor activity to predict the transition from a normal state to an abnormal state.
- *Activity profile* to identify the abnormal activity of the observable states by measuring the sensor deviation from the normal behavior. In short, it characterizes the behavioral signature corresponding to the normal activity of a given subject with respect to a given object.
- *Concept Drift* to measure the change in the user behavior over a period of time.
- *Control Loop* that adapts the checkpoint trigger according to the weighted sum of proportional, average, and derivative sensor measurements over derivative and integral time window.
- *Model* to predict the most probable state based on previous state (normal/abnormal) as well as observed states. This can be accomplished using hidden Markov model (HMM) as described later in this section.

4. Hidden Markov Model

HMM-based approaches correlate the system observations (usage and activity profile) and state transitions to predict the most probable state sequence. It represents a stochastic model of discrete events and a variation of the Markov chain. Like a conventional Markov chain, an HMM consists of a set of discrete states and a matrix $A = a_{ij}$ of state transition probabilities. The states of the HMM can only be inferred from the observed symbols, hence the use of the term hidden. HMM modeling schemes consist of observed (checkpoints) states, hidden (quality of service) states, and HMM (activity) profiles. HMM training using initial data and continuous re-estimation creates a profile that consists of transition probabilities and observation symbol probabilities. Steps involved in HMM modeling include:

- Measuring observed states that are analytically or logically derived from the QoS indicators. These indicators are test-points spread all over the system representing competing risks derived analytically or logically using QoS checkpoint (QC) indicators. Profile deviation can be considered to be a result of several components competing for occurrences of the deviation. In this model QoS checkpoint (QC) engine derives continuous multivariate observation, which is identical to the mean and standard deviation model except that it is based on correlations among several metrics.
- Resource activity trend that corresponds to resource activity monitored over a larger sampling period and represent characteristics that repeat over that sampling period. For example, CPU activity changing depending upon the time of the day. Each period of activity can be thought of as an extra dimension of activity measure.
- Event interval that represents a time period between two successive activities. For example, logging attempts between two consecutive intervals fall in this category.

- Estimating an instantaneous observation probability matrix that indicates the probability of an observation, given a hidden state $p(S_i|O_i)$. This density function can be estimated using explicit parametric model (usually multivariate Gaussian) or implicitly from data via non-parametric methods (multivariate kernel density emission).
- Estimating hidden states by clustering the homogeneous behavior of single or multiple components together. These states are indicative of various QoS states that need to be identified to the administrator. Hidden states $S = \{S_1, S_2, , S_{N?1}, S_N\}$ are the set of states that are not visible but each state randomly generates a mixture of the M observations (or visible states O). The probability of the subsequent state depends only upon the previous state.
- Estimating hidden state transition probability matrix using prior knowledge or random data. This prior knowledge and long-term temporal characteristics are an approximate probability of state components transitioning from one QoS state to another.

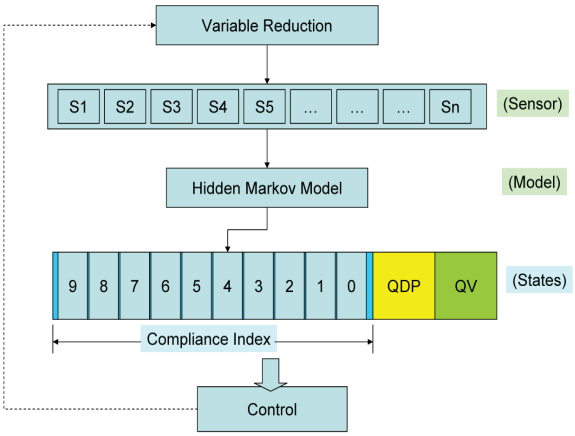


Fig. 3. Hidden States representing the platform policy compliance. These states are estimated from the checkpoints spread over the system in the form of sensors. Checkpoint sensors can be configured for accuracy.

The complete HMM model is defined by the following probabilities: transition probability matrix $A = \{a_{ij}\}$, where $a_{ij} = p(S_i|S_j)$, observation probability matrix $B = (b_i(v_m))$, where $b_i(v_m) = p(v_m|S_i)$, and an initial probability vector $\pi = p(S_i)$. The observation probability represents an attribute that is observed with some probability if a particular failure state is anticipated. The model is represented by $M = (A, B, \pi)$. The transition probability matrix is a square matrix of size equal to the number of states and represents the state transition probabilities. The observation probability distribution is a non-square matrix whose dimension equals the number of states by the number of observable, and represents the probability of an observation for a given state. The PDS has the following states:

- HMM state indicates the anticipated degree of compliance by the platform. This compliance follows the policy that governs the power, thermal and performance (Service Level Agreement) requirements. Therefore HMM state exists as Compliance Index (CI) that ranges from 0-9. While a factor of 9 indicates a stable system that is fully compliant,

a value of 0 indicates a compliant but un-stable system with over(or under) resource allocations, frequent variations and reactive control.

- QoS Deviation in progress (QDP) indicates an activity that is setting itself up and expected to cause the overall quality of service to deteriorate.
- QoS Violation (QV) indicates a successful QoS violation. A successful violation will be accompanied with unusual resource usage (CPU, memory, IO activity, and so on) and a low compliance indicator.

Power, Thermal and performance variations in a system can result in sub-optimal states that may need correction for platform policy compliance. The sub-optimal states need to be predicted well in advance such that corrective actions can be employed within an opportunistic window of time. Such conditions can be predicted using a set of sensors that share probabilistic relationship with the available states. In a platform these sensors are available as activity counters, temperature monitors, power monitors, performance monitors etc.

4.1 CPU power variables

Contribution of CPU power consumption can be measured by calculating fetched micro-operations ((μops)). This metric is directly related to power consumption and also represents the amount of internal paralleism in the processor. On the other hand, instructions retired matric only reflects the useful work and neglects the work done in execution of incorrect branchs and pipeline flushes. The processor decomposes x86 instructions into multiple micro-operations which are then sent to the execution units. Complex instruction mix can result in a false calculation of the amount of computations performed. μops acts independent of complex instructional mix and normalizes this metric to give useful counts.
Apart from using the CPU HW counters, we also use O.S performance meters. One such matrix is CPU utilization that can be used in lieu of μops with a reduced degree of accuracy.

4.2 Memory power variables

It is possible to estimate power consumption in DRAM modules by using the number of read/write cycles and percent of time within the precharge, active and idle states (Janzen, 2001). Since none of these events are visible to the microprocessor, we indirectly estimate them by measuring memory bus accesses by the processor and other events that can be monitored at the CPU. Whenever a memory transaction cannot be satisfied by an L2 cache, it triggers a cache-miss action and performs a cache-line sized access to the main memory. Since the number of main memory accesses is directly proportional to the number of L2 misses, it is possible to approximate memory access count using L2 cache-miss count. In reality the relation is not that simple, but there is still a strong causal relationship between L2 misses and main memory accesses. TLB Misses is another variable that can be significant to power estimation. Unlike cache misses, which mainly cause a cache line transfer from/to memory, TLB misses results in the transfer of a page of data. Due to the large page sizes, they are stored on the disk and hence power is consumed on the entire path from the CPU to the hard disk.

4.3 I/O power variables

Three major indicators of the I/O power are (1) DMA Accesses, (2) Un-Cacheable accesses and (3) Interrupt Activity. Out of these three indicators, Interrupt/cycle is the dominant indicator of the I/O power. DMA indicators perform suboptimal due to presence of various performance enhancements (like write-combining) in the I/O chip. I/O interrupts are

typically triggered by I/O devices to indicate the completion of large data transfers. Therefore, it is possible to correlate I/O power to the appropriate device. Since this information is not available through any CPU counters, they are made available by the operating system (using perfmon).

4.4 Thermal data

Apart from various performance counters defined in previous sections, we also consider using thermal data which available in all the modern components (CPU, Memory) and accessible via PECI BUS.

The heat produced by a component essentially corresponds to the energy it consumes, so we define it as:

$$Q_{component} = P(Util) \cdot Time \tag{8}$$

where, P(Util) represents the average power consumed by the component as a function of its utilization. For most components, a simple linear formulation correctly approximates the real power consumption:

$$P(Util) = P_{base} + Util \cdot (P_{max} - P_{base}) \tag{9}$$

where, P_{base} is the power consumption when the component is idle and P_{max} is the consumption when the component is fully utilized.

5. QoS Checkpoint control

QoS represents a fitness component that maximizes the work-load compliance index by using a minimal amount of resources. The QoS contribution of each element is dependent upon optimal resource allocation that would maximize the CI index and not violate the platform policy (Power Budgeting etc.). A high CI may still demonstrate low QoS due to non-compliant resource distribution. While HMM model predicts the new HMM state, contributing elements predict the desired resource allocation to maximize the CI. CI acts as a feedback path for training purposes that demonstrates the sensitivity of the resource allocation (or de-allocation) throughout the life of the platform. Individual components can build proprietary cost functions (as described below) to predict the desired resource allocation. Once a steady state condition is reached, where the compliance index is sustained by a given set of resources, any deviation from that profile can be construed as a QoS (or profile) violation. QoS indirectly measures the workload compliance efficiency and tries to maximize the compliance factor, as shown in Equations 10 and 11:

$$QoS = CI \cdot \left(1 - \frac{1}{N} \cdot \sum_{i}^{N} R_i\right) \tag{10}$$

$$R_i = \frac{max[0, (R_i^d - R_i^a + \epsilon)]}{R_i^d} \tag{11}$$

where CI represents the compliance index of the workload, R_i^d represents the desired (profiled) resource requirement for component i, R_i^a represents the current resource allocation and N is the number of components that shares that resource. It should be remembered that variations in workload demands may require changing the resource allocation (R_i^d) to maintain the maximum compliance index.

In this section we discuss the applicability of the control theoretic architecture that drives the defensive response based on hysteresis to reduce the incidence of false positives, thereby avoiding inappropriate ad hoc responses. Excessive responses can slow down the system and negatively impact the effectiveness of the PDS. PDS control responses are related to adjusting component functionality (such as throttling), alert generation (to predict QoS violation state) and analyzing concept drift. We introduce checkpoint control loop that acts as the first state of a multistage QoS detection system of sequential PDS. The process output of the control loop provides the observability of an individual QoS checkpoint that aids in the state estimation. Collective observations from several checkpoints are fed into the statistical model (in this case HMM) responsible for predicting the state transition. It is imperative that any such output should be stable and free of oscillations. Response measures are delayed to account for delay involved in the estimation of QoS state based on observations from other checkpoints.

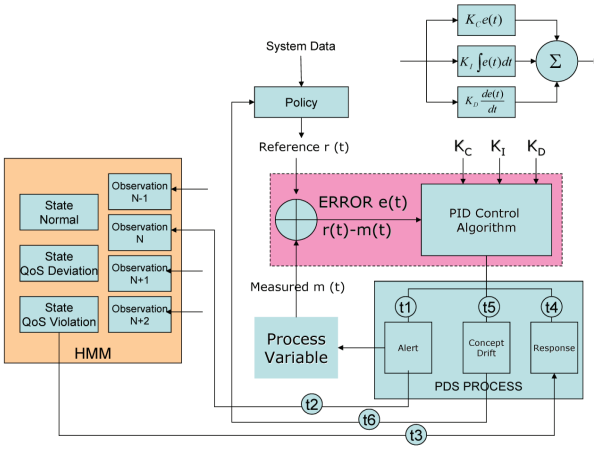


Fig. 4. PID control loop for QoS checkpoint. The Process output (alert) constitutes the observation (emission) in an HMM. A true-positive response is fed back to the process response unit of the PID control to aid runtime retraining. Concept drift analysis aids in re-setting the reference point.

An appropriate response can be built into the QoS checkpoint approach that predicts the QoS divergence pattern and triggers the selective response to a control loop. The PID controller (Fig. 4) may execute one such control loop that takes a measured value from a QoS checkpoint and compares it with a reference value. The difference is then used to trigger alert (abnormal activity) to the process in order to establish the process' measured value back to its desired set-point. It is built with a weighted integral and differential response to the trigger mechanism along with the reactive response to an instantaneous measurement. PID controller can adjust the process outputs based on the history and rate of change of the error signal, which gives more accurate and stable control. This avoids the situation where alerts may not be the true representation of QoS activity due to false positives. Such miscalculations can result in either disproportionate and costly corrective (or defensive) measures or complete failure. The reference (set-points) values are dynamic in nature and set as a part of coarse grain settings that are estimated over long periods of time. These re-estimates are required to account for the changing user behavior, also referred as concept drift. While the set-point

(reference) may remain constant over a long period of time, it can change due to user behavior or system policy driven by a temporary change in the operating environment. System data and the process feedback provide hints that are then used to change the set-point (or set-point weights) in steps based on system policy. System policy is driven by long-term hysteresis based on the system's behavior and the well-known relationship with various checkpoints.

5.1 QoS attributes

Local policy operates within the server node construct and is managed by a management container (manageability Engine, Base Board Management Controller etc.). The goal of local policy is to maximize performance per watt while it operates within total allocated power ($P^t_{alloc}(t)$). In order to implement local policy, the following conditions are desired:

- Ability to accurately monitor power at sub-component granularity.
- Ability to accurately control power at sub-component granularity. Sub-Component power limits are controlled by managing its energy within a given interval. Since energy is equivalent to integrating power over time, power limits are controlled by managing its running averages over time.

$$E^i_T = \int_0^T N_i(t)dt; \;\; N_{i,avg} = \frac{E^i_T}{T} \tag{12}$$

- Ability to accurately monitor average workload performance at runtime.
- Ability to distribute the power among sub-components based on a performance maximization function. This function can be realized using simple random walk or complex evolutionary algorithms.
- Ability to communicate any power or performance credits accumulated in successive evaluation periods.

Felter et al propose a power shifting policy (Felter, 2005) that is analogous to optimal power distribution discussed in this section. This policy reduce peak power consumption by using workload-guided dynamic allocation of power among components incorporating real-time performance feedback, activity-related power estimation techniques, and performance-sensitive activity-regulation mechanisms to enforce power budgets. In addition to the necessary elements required to implement local and global policies, following autonomics infrastructure items are required to build the power distribution model that can be summarized as follows:

5.1.1 Adaptive sampling Infrastructure (LSI)

LSI is responsible for setting the optimal size of the monitoring/control interval. While shorter intervals are better for accuracy, they can overwhelm the natural behavior of the workload. Furthermore, short control intervals impose stronger than necessary constraint of power budgets. Therefore it is desirable to construct the sampling scheme that is statistically proportional to the proximity of the process to the critical threshold according to following equation:

$$T = T_{base}\left(1 + \alpha \cdot \frac{N_{alloc} - N_{avg}}{N_{alloc}} + \beta \cdot \frac{Pr_0 - Pr_{avg}}{Pr_0}\right); \;\; \alpha + \beta = 1 \tag{13}$$

5.1.2 Local Cost Minimization Function (LCMF)

LCMF performs the power distribution amongst node sub-components (DIMM(s), CPU(s), I/O, LAN(s), Storage etc.) in a manner that maximizes the performance while operating under a constant power budget. This function trains itself by utilizing historical trends, identifying repeating patterns. These patterns can be represented in the form of discrete HMM CI states (Sec. 4) that can predict the degree of policy compliance in the future. For example, system sub-components power allocation function is given by:

$$N_{alloc} \geq \sum_{k=1}^{n} (N_k = [f(x_k) + N_{k,min}]) \qquad Pr_o \leq Pr_{avg} \tag{14}$$

$$Pr_{avg} = \sum_{k=1}^{n} C_k \cdot (N_k)^{c_k} \tag{15}$$

Where:
$f(x_k)$ = Power consumed by component k as a function of performance state x_k
$N_{k,min}$= Minimum power of component k
C_k and c_k= Trained coefficients of performance equation. For linear model, $c_k = 1$.
N_{avg} = Average power consumption
N_{alloc} = Node Power Budget allocated by global policy
Pr_o = Desired performance
Once performance model is trained, N_k can be adjusted for maximum performance gain. Discrete states prediction triggers the control action that pro actively mitigates the effects of the performance loss (required to enforce a given policy).

5.1.3 Global Cost Minimization function (GCMF)

GCMF works similar to local cost maximization function on server nodes. It performs the power distribution amongst server nodes in a manner that maximizes the performance while operating under a constant global power budget.

5.1.4 Running Average Power Synthesizer (RAPS)

RAPS is a running average power calculator for a monitored quantity over an enforcement window. RAPS measurement allows for all the sub-components and server nodes to control average power over a given interval.

5.1.5 Running Average Performance Synthesizer (RAPrS)

RAPrS is a running average performance calculator of the node workload. This is monitored for each individual workload running in a server node.
Running average power and performance calculator can be utilized to dynamically monitor the power and performance trends over an adjustable evaluation window of time and maintain the average power at or below a given threshold.

6. Profile deviation detection (PDS) architecture

This section characterizes components of the PDS that cooperate with each other to predict a QoS noncompliance (violation) state. PDS is deployed as a part of an autonomic element (AE) that detects the signs of QoS violation locally and independent of other AEs.

The PDS architecture comprises multiple stages with an information feedback mechanism between stages. These stages can be roughly defined as follows:

- QoS Checkpoint Control Stage (QCCS) is the observability stage with an objective to produce stable emissions using continuous estimations. This stage is also responsible for detecting temporary changes due to legal activity and concept drift signifying changing long-term application behaviors. This decision is crucial because a drift in the normal behavior may also be falsely predicted a QoS violation. Observation can be rejected as a noise, or classified to a valid state based on the trending, similarity between unclassified states tending toward certain classification, and feedback from state machine based on other independent observations.
- QoS State Detection Stage (QSDS) receives the observability data from multiple checkpoints and predicts the transition to one of the hidden states (normal, QoS violation) based on trained statistical model. An estimated QoS decision is fed back to QCCS, which helps re-estimating the usage trends while avoiding any false positive preemptive responses.
- QoS Response Stage (QRS) is responsible for initiating the corrective (healing) actions due to state transition. These actions may scale back any abnormal activity as seen in the observability data. A mis-predicted state transition may initiate an inappropriate response and will have negative effect on checkpoint activity.

After various components of the model are trained, it enters a runtime state where it examines and classifies each valid observation. Various components of a QoS detection system are explained in the following sub-sections.

7. QoS checkpoint control stage (QCCS)

QCCS represents the feedback control component (Figure 5) for an individual QoS checkpoint. It comprises a measurement port, PID controller, observation profiler, concept drift detector (CDD), and feedback path to the process input.

Measurement Port is composed of fast-acting software and silicon hooks that are capable of identifying, counting, thresholding, time-stamping, eventing, and clearing an activity. Examples of such hooks are performance counters, flip counters (or transaction counters), header sniffers, fault alerts (such as page faults), bandwidth usage monitors, session activity, system call handling between various processes and applications, file-system usage, and swap-in/swap-out usage. Measured data is analyzed as it is collected or subsequently to provide real-time alert notification for suspected deviant behaviors. These fast-acting hooks are clustered to enact an observation. Measurements can be sampled at regular intervals or cause an alert based on a user-settable threshold.

Observation Profiler monitors various inputs for maintaining/re-estimating activity profile that ascertains a rough (partially perfect) boundary between normal and abnormal activity and characterized in terms of a statistical metric and model. A metric is a random variable representing a quantitative measure accumulated over a period. Measurements obtained from the audit records when used together with a statistical model analyze any deviation from a standard profile. An observation profiler receives multiple feedbacks from PID control output, event trigger and QSDS, and performs recursive estimations that generate successive probabilistic profile data estimates with a closed-form solution. A trigger event is generally followed by a change in the PID control output that initiates a recovery response. A true

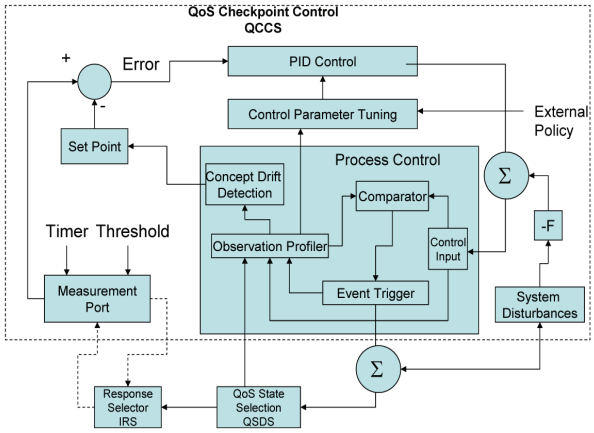


Fig. 5. QCCS is responsible for providing the stable observability data to the QoS state detection stage. This data is profiled for variances due to changing user behavior and temporary changes in system environment (also referred to as disturbances).

positive recovery response will scale back the checkpoint activity to normal. A false positive action will instead cause oscillations, degraded system performance, or little change in the measured error. Activity profile data consists of probability distribution function (pdf) and the related parameters (e.g. variance, mean, activity drift factor etc). Successive observations are evaluated against this profile which results in its new profiles and drift detection. An observation (emission) can also be a set of correlated measurements but represented by a single probability distribution function. Each of these measurements carries different weights as in multivariate probability distribution. Such relationship is incorporated into the profile for the completeness of the observation and reduces the dimensionality for effective runtime handling. Observability in this case is derived out of the profile that represents a consolidated and single representation of activity. A sample profile data structure is defined as follows.

NFS Profile {

Observation Name = NFS Activity

Input Events = $\{\text{Disk I/O}, \text{Network I/O}, \cdots\}$

Output Emissions = Function (Input Events)

PDF Parameter = $\{D[N], D[HI], D[FI], D[IP], D[IS]\}$

Unclassified Observation = $\{U[t_1], U[t_2], \cdots, U[t_n]\}$

Concept Drift Data = $\{\eta_{t1}, \eta_{t2}, \cdots\}$

}

Concept Drift Detector detects and analyzes the concept drifting (Widmer, 1996) in the profile where training data set alone is not sufficient, and the model (profile) needs to be updated continually. When there is a time-evolving concept drift, using old data unselectively helps if the new concept and old concept still have consistencies and the amount of old data chosen arbitrarily just happen to be right (FAN, 2004). This requires an efficient approach to data mining that helps select a combination of new and old (historical) data to make an accurate re-profiling and further classification. The mechanism used is the measurement of Kullback-Leibler (KL) divergence (Kullback, 1951), or relative entropy measures the kernel

distance between two probability distributions b(.|.) of generative models as expressed in Equation 16:

$$\alpha_t = KL(b(v|\theta'_t), b(v|\theta_t)) \tag{16}$$

Where: α_t = KL divergence measure θ'_t = New Gaussian component θ_t = Old Gaussian component at time . v = Observation vector We can evaluate divergence by a Monte Carlo simulation using the law of large numbers (Grimmett, 1992) that draws an observation θ'_t from the estimated Gaussian component , computes the log ratio, and averages this over M samples as shown in Equation 7.7 as:

$$\alpha_t \approx \frac{1}{M} \sum_{i=1}^{M} \log \left(\frac{b(v_i|\theta'_t)}{b(v_i|\theta_t)} \right) \tag{17}$$

KL divergence data calculated in the temporal domain are used to evaluate the speed of the drift, also called drift factor ( ). These data are then used to assign weights to the historical parameters that are then used for re-profiling.

Feedback Path is responsible for feeding back the current state information to the profile estimator. The current state information is calculated by running the ISDS module using the current model parameters. This information is then used by the profiler to filter out any noise and re-estimate the activity profile data. If a trigger event is not followed by a state transition, then a corrective action is performed to minimize the false positives in the future.

PID Controller (Fig. 4) generates an output that initiates a corrective response applied to a process in order to drive a measurable process variable toward a reference value (set point). It is assumed that any QoS activity will cause variations in the checkpoint activity, thereby causing a large error. Errors occur when a disturbance (QoS violation) or a load on the process (changes in environment) changes the process variable. The controller's mission is to eliminate the error automatically. A discrete form of PID controller is represented by Equation 18:

$$u(nT) = P + I + D + u_0 \tag{18}$$

where,

$$\begin{aligned} P &= K_p \cdot e(nT) \\ I &= K_i \cdot T \cdot \sum_{i=(nT-w)}^{nT} e(i) \\ D &= K_d \cdot \frac{e(nT) - e(nT-1)}{T} \end{aligned}$$

where e(t) is the error represented by difference between measured value and set-point, w is the integral sampling window, nT is the n-th sampling period, and K_p, K_i and K_d are the proportional, integral, and derivative gains respectively. Stability is ensured using the proportional term, the integral term permits the rejection of a step disturbance, and the derivative term is used to provide damping or shaping of the response. While integral response measures the amount of time the error has continued uncorrected, differential response anticipates the future errors from the rate of change of error over a period of time. The desired closed-loop dynamics are obtained by adjusting these parameters iteratively by

tuning and without specific knowledge of a QoS detection model. Control parameters are continuously tuned to ensure the stability of the control loop in a control-theoretic sense, over a wide range of variations in the checkpoint measurements. While control parameters are evaluated frequently, they are updated only when improvement in stability is anticipated. These updates can be periodic over a large period of time.

7.1 Relevant profiles

This section look into events that forms input to the profile structure. Exploiting temporal sequence information of event leads to better performance (Ghosh, 1999) of profiles that are defined for individual workloads, programs, or classes. An abnormal activity in any of the following forms is an indicator of a QoS variation:

- CPU activity is monitored by sampling faults, inter-processor interrupt (IPI) calls, context switches; thread migrations, spins on locks, and usage statistics.
- Network activity is monitored by sampling input error rate, collision rate, RPC rejection rate, duplicate acknowledgments (DUPACK), retransmission rate, timeout rate, refreshed authentications, bandwidth usage, active connections, connection establishment failure, header errors and checksum failures and so on.
- Interrupt activity is monitored by sampling device interrupts (non-timer interrupts).
- IO utilization is monitored by sampling the I/O requests average queue lengths and busy percentage.
- Memory activity is monitored by sampling memory transfer rate, page statistics (reclaim rate, swap-in rate, swap-out rate), address translation faults, pages scanned and paging averages over a short interval.
- File access activity is monitored by sampling file access frequency, file usage overflow, and file access faults.
- System process activity is monitored by sampling processes with inappropriate process priorities, CPU and memory resources used by processes, processes length, processes that are blocking I/Os, zombie processes, and command and terminal that generated the process.
- System faults activity represents an illegal activity (or a hardware error) and is sampled to detect abnormality in the system usage. While rare faults represent a bad programming, but spurts of activity indicate an attack. n System calls activity measures the system-call execution pattern of a workload. It is used to compare runtime system-call execution behavior with the expected pattern and detect any non-expected execution of system calls. Pattern-matching algorithms match the real-time sequence of system-calls and predict a normal or abnormal behavior (Wespi, 1999).
- Session activity is monitored by sampling the logging frequency, unsuccessful logging attempts, session durations, session time, session resource usages, and so on.
- Platform Resource Activity is monitored by sampling CPU, DIMM, I/O power consumption, and thermal data. Additionally CPU, memory, and I/O bandwidth performance can also be measured using performance counters available within the CPU core or un-core logic.

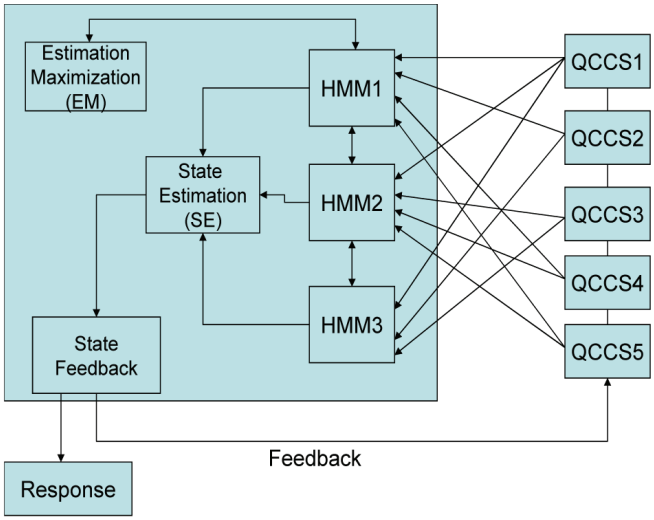


Fig. 6. QoS state detection stage (QSDS)

7.2 QoS State Detection Stage (QSDS)

QSDS defines the statistical model that is responsible for predicting the current QoS state based on observable data inputs received from QCCS modules. In this context we may choose HMM where states are hidden and indirectly evaluated based on model parameters. QCCS trigger output acts as an emission to a specific HMM model and weighted according to its significance relative to that model. HMM emissions are defined as processed observation, derived from one or more temporal input events using a processor function. They represent competing risks derived analytically or logically using checkpoint indicators. Platform states can be considered to be a result of several components competing for the occurrences of the anomaly. Observed inputs may be expressed as a weighted fraction of individual observations from multiple checkpoints in an attempt to enhance the performance of QoS state detection. Similar observed inputs (emissions) may be distributed among mixture of models, usually Gaussian, with weights given to each model based on trivial knowledge and continuous training. This approach is advantageous as it allows one to model the QoS states at varying degree of granularity while retaining the advantages of each model. Depending upon the data characteristics (amount of data, frequency); models can be adapted by modifying weights such that complex models are favored for complex inputs and vice versa. Such mixture model can be represented as Equation 19:

$$p(v) = \sum_{k=1}^{K} a_k b(v|\theta_k) \tag{19}$$

$$a_k > 0 \ \ and \ \ \sum a_k = 1$$

Where: v = Observation vector $p(v)$ = Modeled probability distribution function a_k = Mixture proportion of component K = Number of components in the mixture model θ_k = Distribution Parameters of component $b(v|\theta_k)$ = Distribution function for component

Figure 6 illustrates the HMM-x sub-block which is responsible for receiving the abnormal activity alert and processes the interrupt to service the hidden-state (QoS) estimation. It

maintains the HMM data and interacts with the expectation-maximization (EM) block and the state-estimation (SE) block for retraining and state-prediction flows. This block also implements reduced dimensionality by combining multiple inputs into a single observation with its own probability distribution function. This observation is then fed into the EM and SE block for state estimation. The EM algorithm Grimmett (1992) provides a general approach to the problem of maximum likelihood (ML) parameter estimation in statistical models with variables that are not observed. The evaluation process yields a parameter set which it uses to assign observations points to new states. The EM sub-block is responsible for finding the ML estimates of parameters in the HMM model as well as mixture densities (or model weights) and relies on the intermediate variables (also called latent data) represented by state sequence. EM alternates between performing an E-step, which computes an expectation of the likelihood, and an M-step, which computes the ML estimates of the parameters by maximizing the expected likelihood found on the E-step. The parameters found on the M-step are then used to begin another E-step, and the process is repeated. In the HMM mixture modeling, QoS checkpoint events under consideration have membership in one of the distributions we are using to model the data. The job of estimation is to devise appropriate parameters for the model functions we choose, with the connection to the data points being represented as their membership in the individual model distributions. SE is responsible for modeling the underlying state and observation sequence of HMM mixture to predict state sequences for new QoS states using the Viterbi algorithm (to find the most likely path through the HMM). Trained mixture appears to be a single HMM for all purposes and can be applied as a standard HMM algorithm to extract the most probable state sequence given a set of observations. Estimates for the transition and emission probabilities are based on multiple HMM models and are transparent to the standard HMM models. The Viterbi algorithm is a dynamic algorithm requiring time $O(TS^2)$ is the number of time steps and S is the number of states) where at each time step it computes the most probable path for each state given that the most probable path for all previous time steps has been computed. The state feedback sub-block feeds back the estimated state to the observation profiler in ICCS, which then uses this data for recalibrating the profile.

8. System considerations

Profile detection in local platform components (CPU, DIMM etc.) is limited to profiling local activity using floating QCCS modules. The intent is to reduce the system complexity and enhance the likelihood of software reuse. These hooks exist to accelerate the combined measurements of the clustered components with an ability to send alerts based on a systems-wide policy. It contains the hardware and software that act as a glue between transducers and a control program that is capable of measuring the event interval and event trend with an ability to generate alerts upon departure from normal behaviors (represented by system policy). In this specific case, the feedback control loop is implemented partially in the silicon (QCCS block) with configurable control parameters. To further enhance the auto-discoverability, modularity, and re-usability, configuration and status registers may be mapped into the capability pointer of the PCI Express configuration space. Similar mechanisms exist today in the very basic form as performance counters (PerfMon), leaky-bucket counters, and so on. These counters need to be coupled with QCCS modules that contain a PID controller, profilers, threshold detectors, drift detectors, and coarse-grain tuners. QCCS modules should be implemented in isolation from the measured components such that a single QCCS component can multiplex between multiple measurement modules.

While some of the checkpoints are used for local consumption, others are shared with the monitor nodes to aid in cooperative state estimation. These checkpoints share the trigger data with the monitor nodes and contribute as the node's contribution to the mixture of HMM. The enormous amount of measurement data and computational complexity are a paramount consideration in the design of an effective PDS system.

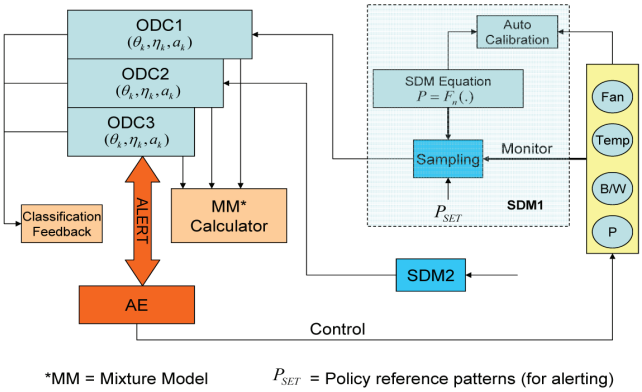


Fig. 7. Illustration of the relationship between events (circles), sensors (SDM) and classifiers (ODC). Clusters of events (marked by similar colors) are registered to an SDM. SDM upon evaluating the event properties, generate an event to ODC. ODC is responsible for classification, trend analysis and drift calculation.

8.1 Sensor data measurement (SDM)

SDM hooks reduce the system complexity and increase the likelihood of software reuse. SDM accelerates the combined measurements of the clustered components with an ability to send alerts using a systems policy. Hardware and software acts as glue between transducers and control program that is capable of measuring the event interval, event trend with an ability to generate alerts on deviation from normal behavior (represented by system policy). The SDM hardware exists as a multiple-instance entity that receives alert vectors from various events spread all over the system. A set of correlated events form a cluster and are registered against a common SDM instance. This instance represents the Bayes optimal decision boundaries between set of pattern classes with each class represented by an SDM instance and associated with a reference vector. Each SDM instance is capable of trending and alerting and integrates the measurements from the event sensors into a unified view. Cluster trending analysis is unusually sensitive to small signal variations and capable of detecting the abnormal signals embedded in the normal signals by supervised learning (Kohonen, 1995).

As illustrated in fig. 7, policy based reference patterns are manually (or automatically) identified that would result in alerts to ODC. Simple patterns may be represented in a form of RAW thresholds. More complex patterns would require statistical processing of the RAW data into meaningful information which is then matched against reference patterns.

8.2 Observation Data Classifier (ODC)

ODC hooks accelerate the classification of an observation alert generated by SDM. This is multiple-instance hardware (Figure 7) capable of handling multiple observations in parallel.

Each registered observation instance of the ODC hook consists of probability distribution parameters of each state. Upon receiving an SDM alert, the observation corresponding to this alert is then classified to a specific state. Reclassification of observed data may cause changes in the probability distribution parameters corresponding to the state. ODC is capable of maintaining the historical parameters, which are then used to calculate concept drift properties (drift factor, drift speed, and so on) using drift detector.

8.3 Mixture Model (MM) Calculator

The MM calculator determines the probability of the mixture (usually Gaussian) for each state, using the current observation. During the system setup, event vectors are registered against SDM instance. These events are clustered and processed in its individual SDM. The processing includes trigger properties that initiates an observation. These observations then act as single-dimensional events that are registered to its ODC. Upon receiving the trigger, ODC performs reclassification of the observation (derived from the trigger) and calculates the concept drift. It should be noted that this hardware is activated upon a trigger by its parent.

9. Summary

Since the QoS state resulting from service-level-agreements (SLA) compliance cannot be inferred directly by monitoring any specific parameters, we need to predict QoS violations based on a mixture of observable data points, events, and current states. This leads to a statistical mechanism for QoS prediction using HMM where observed data are represented as a weighted mixture component. Using this mechanism, an observed deviation from a normal behavior carries a higher probability of being in a sub-optimal state. We define HMM based statistical model that predicts the QoS state based on observable data inputs received from checkpoint control modules. This chapter also introduces the concept of a feedback control mechanism that regulates the defensive response to every perceived sub-optimality (or abnormality). As explained earlier, this helps reduce the false positive rate, which is one of the major problems in Profile Detection System (PDS). Modern silicon (CPU, I/O hubs, PCI Express devices) contains performance counters that can be measured at moderate granularity. To avoid software overhead, these counters can be mapped to the feedback control modules. Various functional units of the silicon should be able to profile the activity trends supported by the eventing mechanism in a power efficient manner. Physical layer design should support protocols related to optimal monitor-node selection based on user-defined policies and authentication. PDS needs to understand relationships, relevance, and correlation between multiple triggers (or emissions) in a computationally efficient manner.

10. References

Denning, Dorothy E. (1987). An Intrusion-Detection Model, *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, VOL. SE-13, NO. 2, FEBRUARY 1987, 222-232.

Fan, W. (2004). Systematic data selection to mine concept-drifting data streams. *ACM SIGKDD*, 2004.

Felter, W.; Rajamani, K.; Keller, T.; and Rusu, C. (2005). A performance-conserving approach for reducing peak power consumption in server systems. In Proceedings of the *19th Annual international Conference on Supercomputing*, Cambridge, Massachusetts, June 20 - 22, 2005.

Ghosh, A. K.; Schwartzbard, A. and Schatz, M. (1999). Learning program behavior profiles for intrusion detection. In Proc. *Workshop on Intrusion Detection and Network Monitoring*, pp. 51-62, Santa Clara, USA, Apr. 1999.

Grimmett, G. R. and Stirzaker, D. R. (1992). Probability and random processes. *Oxford, U.K.: Clarendon Press*, 2nd edition, 1992.

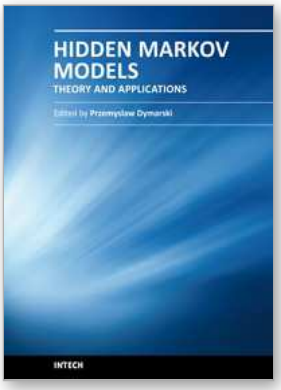
Janzen, J. (2001). Calculating Memory System Power for DDR SDRAM. *Micro Designline*, Volume 10, Issue 2, 2001.

Kohonen, T. (1995). Self-organizing maps. *Springer Press*, 1995.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, vol. 22, pp. 79-86, Mar. 1951.

Wespi, A.; Debar, H. and M. Dacier (1999). An intrusion-detection system based on the Teiresias pattern-discovery algorithm *Eicar'99, Aalborg*, Denmark, Feb. 27-Mar. 2, 1999.

Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drifting and hidden contexts. *Machine Learning*, vol. 23, pp. 69-101, 1996.



Hidden Markov Models, Theory and Applications

Edited by Dr. Przemyslaw Dymarski

ISBN 978-953-307-208-1
Hard cover, 314 pages
Publisher InTech
Published online 19, April, 2011
Published in print edition April, 2011

Hidden Markov Models (HMMs), although known for decades, have made a big career nowadays and are still in state of development. This book presents theoretical issues and a variety of HMMs applications in speech recognition and synthesis, medicine, neurosciences, computational biology, bioinformatics, seismology, environment protection and engineering. I hope that the reader will find this book useful and helpful for their own research.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Rahul Khanna, Huaping Liu and Mariette Awad (2011). Control Theoretic Approach to Platform Optimization using HMM, Hidden Markov Models, Theory and Applications, Dr. Przemyslaw Dymarski (Ed.), ISBN: 978-953-307-208-1, InTech, Available from: http://www.intechopen.com/books/hidden-markov-models-theory-and-applications/control-theoretic-approach-to-platform-optimization-using-hmm

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821