Complexity reduction for Gaussian Process Regression in spatio-temporal prediction

Dinh-Mao Bui*, Thien Huynh-The*, Sungyoung Lee*, YongIk Yoon†
*Computer Engineering Department, Kyung Hee University, Suwon, Korea
{mao, thienht, sylee}@oslab.khu.ac.kr
†Department of Multimedia Science, SookMyung Women's University, Seoul, Korea
{yiyoon}@sm.ac.kr

***Abstract*—To deal with inference and reasoning problems, Gaussian process has been considered as a promising tool due to the robustness and flexibility features. Especially, solving the regression and classification, Gaussian process coupling with Bayesian learning is one of the most appropriate supervised learning approaches in terms of accuracy and tractability. Unfortunately, this combination tolerates high complexity from computation and data storage. Obviously, this limitation makes Gaussian process ill-equipped to deal with the systems requiring fast response time. In this paper, the research focuses on analyzing the performance issue of Gaussian process, developing a method to reduce the complexity and implementing to predict CPU utilization, which is used as a factor to predict the status of computing node. Subsequently, a migration mechanism is applied so as to migrate the system-level processes between CPU cores and turn off the idle ones in order to save the energy while still maintaining the performance.**

Keywords*—Proactive prediction, Bayesian learning, Gaussian process, energy efficiency, CPU utilization.

I. Introduction

Gaussian process exists in many research fields, such as data communication, networking and computer science. It is widely used as a non-parametric and probabilistic approach to model the characteristics of a target system. However, this technique does have one significant drawback. In a standard implementation, Gaussian process costs $O(n^3)$ for computational complexity and $O(n^2)$ for storage complexity, when calculating n training points of a dataset. Theoretically, these results come from the matrix inversion and the log determinant calculation for the covariance matrix and degrades the overall performance.

There has been quite a bit of research conducted aimed at overcoming these limitations, which focus on utilizing well-known mathematical methods, like Cholesky decomposition and reduced rank covariance matrix. Nevertheless, it is still not fast enough to satisfy the strict requirements of a large-scale system. Others are optimization techniques, which seem to be effective. Nevertheless, these techniques also have reliability issues and are still slow compared with mathematical methods.

In this paper, we propose a solution for the complexity problem in Gaussian process regression, particularly related to time series prediction in the periodic spatial-temporal dimension. Theoretically, this approach constructs a combination of mathematical modeling, convex optimization to reduce the complexity. In order to verify the proposed idea, an application aimed at proving the effectiveness of the proposed method is devised. Basically, this application predicts the multi-cores CPU utilization and issues the process migration between the cores in order to achieve the overall energy efficiency while maintaining a high performance.

This paper is organized as follows. In Section 2, we provide a summary of the related works that are relevant to this topic. We detail the regression framework in Section 3. In Section 4, we conduct the performance evaluation of the energy saving application. Our conclusion and discussion are summarized in Section 5.

II. Related Works

In research concerning the use of Gaussian process for predicting chaotic time series [1], the Evolving Gaussian process method has been proposed to identify chaotic time series events in the system, as an on-line method for Gaussian process modeling. In this method, the information is received in the streaming mode and the hyper-parameters are then adjusted, adapting the prediction model to the data. Studying hyper-parameters on-line is interesting; however, in the optimization phase, this method only engages the Conjugate Gradient (CG) and Cholesky decomposition for hyper-parameter learning, which decreases the overall performance because of the high computational complexity.

Other research on Gaussian process for Big Data [2], focused on the Stochastic Variational Inference (SVI) method for constructing Gaussian process models. One interesting advantage of this approach is the independent complexity with regard to the dataset. This method allows variational inference in a very large dataset, showing impressive performance in terms of the speed of inference. Nonetheless, the SVI only works properly in a special system in which the observational and latent variables are globally factorized. Unfortunately, Gaussian process does not have global variables. In addition, because of their input independence properties, the SVI-oriented methods are affected by large variances, which results in less precise solutions.

Another approach using Improved Fast Gauss Transform [4], which is a Matrix-Vector Multiplication method, performed better in the hyper-parameter learning. This approach improves upon the Fast Gauss Transform, which is originally derived from the well-known Fast Multipole Method, by adaptively choosing the precision parameters during the approximation process. In addition, the IFGT engages a divide-and-conquer mechanism by partitioning the domain with k-centers

clustering and caching the sum contribution in each level for acceleration purpose. Despite these notable improvements, the IFGT still has a critical drawback in that this method mostly relies on the Fast Gauss Transform, which is only effective if the objective function can be expanded, using the Taylor series expansion, to Gaussian-type potential. Unfortunately, this requirement is not always satisfied in the hyper-parameter learning phase, resulting in unreliable calculations for the hyper-parameters estimation. Further elaboration of this issue and potential solutions will be addressed later in this research.

Although there has been a significant amount of research focusing on Gaussian process regression, not much progress has been made in terms of performance improvement, especially related to optimization. Furthermore, due to the fact that large-scale data is popular in every current distributed system, it is necessary to develop a low-complexity and reliable method to process this kind of data and give the predictive information.

III. Proposed Method

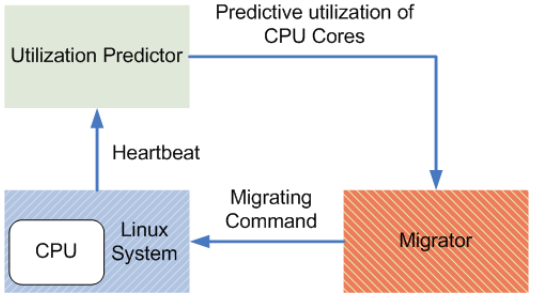


Fig. 1: The architecture of energy saving application.

A. Target process

To optimize Gaussian process regression in the spatial-temporal dimension, we choose to use an energy saving application for demonstration. The energy efficiency architecture for a multi-cores CPU is proposed in this section and described in detail in Figure 1. Basically, the purpose of this architecture is to pro-actively reduce the energy consumption of the CPU cores. Thus, the main functionality of this application is to empty the workload of the CPU cores and then deactivate or stand-by the idle cores to save energy. To do this effectively, the migration procedure on the Migrator component needs the predictive information of the CPU core utilization to determine the source and the destination in order to migrate the target process (from this point, the system process being considered migration will be known as the target process). Primarily based on the current value of the CPU core's utilization, known as the heart-beat, the predictive utilization is calculated in the Utilization Predictor component and plays a critical role in subsequent decision making. For a fast reaction in term of rapid change in the utilization of the CPU cores, the energy-saving application is implemented as a background daemon application in each computing node and be responsible for migrating the system processes between the local CPU cores.

B. Prediction model

In our application, the object of the prediction is to anticipate the utilization of the CPU cores. Bayesian learning and Gaussian process regression are employed as the inference technique and probability framework, respectively. Because the input data for this model is the time series utilization, curve-fitting is preferred over function mapping for the mapping approach. It is important to note that the curve-fitting is more flexible with regard to the time series data and non-stationary model.

Assuming that the input data is a limited collection of time location $x = [x_1, x_2, x_3, \cdots x_n]$, a finite set of random variable Y=[Y_1, Y_2, Y_3,$\cdots$ Y_n] represents the corresponding joint Gaussian distribution of incoming processes with regard to the time order. This set over the time constraint actually forms up the Gaussian process:

$$f(y|x) \sim \mathcal{GP}\big(m(x), k(x, x')\big) \tag{1}$$

with

$$m(x) = \mathbb{E}\big(f(x)\big) \tag{2}$$

$$k(x, x') = \mathbb{E}\Big(\big(f(x) - m(x)\big)\big(f(x') - m(x')\big)\Big) \tag{3}$$

in which, $m(x)$ is the mean function, evaluated at the location x variable, and $k(x, x')$ is the covariance function, also known as the kernel function. By definition, the kernel function is a positive-definite function, used to define the prior knowledge of the underlying relationship. Basically, the kernel function is only a mandatory requirement when there is a lack of finite dimensional form of the feature space. Otherwise, it can be dropped by directly calculating the sample. However, this feature space dimension is frequently infinite, which means that the kernel function cannot be directly calculated. For this reason, the kernel function technique is often chosen to tackle the Gaussian process regression. In addition, the kernel function comprises some special parameters that specify its own shape. These parameters are referred to as hyper-parameters. Because the input data comes to the Predictor as a set of locations, the kernel should be engaged in matrix form.

$$\mathbf{K} = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{pmatrix} \tag{4}$$

Generally, the Square-Exponential (SE) kernel, also known as the Radial Basis Function (RBF) kernel, is chosen as a basic kernel function. In reality, the SE kernel is favored in most Gaussian process applications, because it requires the calculation of only a few parameters. Moreover, there is a theoretical reason to choose this method, as it is a universal kernel appropriate for any continuous function when enough data is given. The formula for SE kernel is described as follows:

$$k_{SE}(x, x') = \sigma_f^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right) \tag{5}$$

in which, σ_f is an output-scale amplitude and l is a time-scale of the variable x from one moment to the next. l also stands for the bandwidth of the kernel and, thereby, the smoothness of the function in the model. In addition, l also plays the role of judgment for Automatic Relevance Detection (ARD), to discard the irrelevant input dimension. In the next step, we

evaluate the posterior distribution of the Gaussian process. Assuming that the incoming value of the input data is (x_*, y_*), the joint distribution of the training output is y, and the test output is y_*, as below:

$$p\left(\begin{bmatrix} y \\ y_* \end{bmatrix}\right) = \mathcal{GP}\left(\begin{bmatrix} m(x) \\ m(x_*) \end{bmatrix}, \begin{bmatrix} \mathbf{K}(x,x') & \mathbf{K}(x,x_*) \\ \mathbf{K}(x_*,x) & \mathbf{K}(x_*,x_*) \end{bmatrix}\right) \quad (6)$$

here, $\mathbf{K}(x_*, x_*) = k(x_*, x_*)$, $\mathbf{K}(x, x_*)$ is the column vector made from $k(x_1, x_*), k(x_2, x_*) \cdots, k(x_n, x_*)$. In addition, $\mathbf{K}(x_*, x) = \mathbf{K}(x, x_*)^T$ is the transposition of $\mathbf{K}(x, x_*)$. Subsequently, the posterior distribution over y_* can be evaluated with the below mean m_* and covariance C_*.

$$m_* = m(x_*) + \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}(y - m(x)) \quad (7)$$

$$C_* = \mathbf{K}(x_*, x_*) - \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}\mathbf{K}(x, x_*) \quad (8)$$

Then

$$p(y_*) \sim \mathcal{GP}(m_*, C_*) \quad (9)$$

The best estimation for y_* is the mean of this distribution:

$$\overline{y}_* = \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}y \quad (10)$$

C. Hyper-parameter learning phase

The proposed model invokes a set of hyper-parameters $\theta = [\sigma_f, l]$, which exists both in covariance and mean function. Theoretically, these hyper-parameters are supposed to be evaluated through the marginalization process. By using Bayes' rule, the equation (9) can be re-written as shown below.

$$p(y_*|y) = \frac{\int p(y_*|y,\theta)p(y|\theta)p(\theta)\,\mathrm{d}\theta}{\int p(y|\theta)p(\theta)\,\mathrm{d}\theta} \quad (11)$$

In this equation, the marginal likelihood $p(y) = \int p(y|\theta)p(\theta)\,\mathrm{d}\theta$ is the main point of interest. So far, it is clear that the Maximum A Posteriori (MAP) estimation of θ can be obtained when $p(\theta|y)$ reaches its maximum [6]. From the first moment of the inference process, the prior $p(\theta)$ must be assigned for the hyper-parameter to reflect the domain knowledge. Based on the input, this task is not an obstacle. In addition, according to Bayes' rule, the probability $p(\theta|y)$ is known to be proportional to $p(y|\theta)$. Then, the optimization step only involves maximizing the $\log p(y|\theta)$ or minimizing the negative $\log p(y|\theta)$.

$$-\log p(y|\theta) = \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} + \frac{1}{2}\log|\mathbf{K}| + \frac{n}{2}\log(2\pi) \quad (12)$$

Due to the high complexity of re-calculating the matrix inversion, this is a must to consider another method for hyper-parameter learning. Instead of putting effort in minimizing the negative log likelihood, this heavy-load-job can be done faster by approximately minimizing the upper bound of this term [8]. Analytically, in equation (12), the dominant computation focuses on two terms: the data-fit term [9] which is denoted by $\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y}$ and the complexity penalty term or the log determinant $\log|\mathbf{K}|$. Before going further, the equation (12) should be simplified to reduce the complexity. To do that, a good approach is to engage the law of log determinant of the sample covariance matrix which originates in [10]. In this research, by calculating the log determinant $\hat{L}$ of sample covariance matrix $\hat{\mathbf{K}}$, the equation (12) is simplified to

$$-\log p(y|\theta) = \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} + \frac{1}{2}\hat{L} + \frac{n}{2}\log(2\pi) \quad (13)$$

When the process runs for a long time, the term $\hat{L}$ also converges to a constant. This convergence leads to a conclusion that minimizing the negative log marginal likelihood in this domain can only involve minimizing the following reduced negative marginal log likelihood (rMLE).

$$-\log p(y|\theta)_{rMLE} = \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} \quad (14)$$

Traditionally, dealing with this task intuitively concerns inversing the covariance matrix $\mathbf{K}$. This matrix operation normally costs $O(n^3)$ which is very computationally expensive. Motivated by the application in the Stochastic Frontier Model [6] and Kriging problem [11], the Fast Fourier Transform (FFT) can be a promising method to deal with this complexity. As mentioned before, the kernel function is a positive-definite function, then the Fourier Transform makes it possible to transform this kernel to bring the computation from the spatial-temporal domain into frequency domain. Consequently, the most expensive subsequent task is not the matrix inversion. It concerns calculating the power spectrum which costs only $O(nlogn)$. Obviously, this cost is much better and faster than the aforementioned traditional approach. To achieve the above advantage, first the Squared Exponential kernel $k_{SE}(x, x')$ in equation (5) needs to be re-written in Fourier Transform representation [12] as shown below.

$$\mathcal{F}_{SE}(\omega) = l\sigma_f^2\sqrt{2\pi}exp(-2\pi^2\omega^2l^2) \quad (15)$$

To accelerate the optimization, the non-uniform Fast Fourier Transform (NUFFT) is applied. According to the rMLE minimization, assuming that $\mathbf{\Phi}$ is the function generates $\tilde{\mathbf{K}} = \mathbf{K}^{-1}$. Under the periodic assumption, the Parseval theorem [11] can be applied to derive the Fourier Transform for equation (14)

$$\mathcal{F}_{rMLE}(\theta) = \mathcal{F}\big(-\log p(y|\theta)_{rMLE}\big) = \frac{1}{2n}\hat{\mathbf{y}}^T\widehat{\mathbf{\Phi} * \mathbf{y}_\circ} \quad (16)$$

in which the hat sign denotes a Fourier Transform and $\mathbf{y}_\circ$ denotes the data vector in the periodic domain. In the next step, by continuing applying the convolution theorem with regard to the constraint $\Phi\mathcal{F}_{SE} \equiv 1$, the final form of the Fourier Transform of the rMLE can be represented as shown below.

$$\mathcal{F}_{rMLE}(\theta) = \frac{1}{2n}\sum_i \hat{\Phi}_i * \hat{y}_i^2 = \frac{1}{2n}\sum_i \frac{\hat{y}_i^2}{\mathcal{F}_{SE}(\omega_i)} \quad (17)$$

With this form of equation (17), the set of hyper-parameters θ is no longer expensive to discover by using some gradient-based techniques. In this case, the Stochastic Gradient Descent (SGD) is chosen because this optimization technique is suitable for the large data set, faster than other gradient descent and critically less sensitive to the local minima [13]. To integrate the Stochastic Gradient Descent into hyper-parameter learning, the partial derivatives of the equation (17) are required for the calculation with regard to each hyper-parameter. These equations are given by

$$\frac{\partial}{\partial l}\mathcal{F}_{rMLE} = \hat{y}_i^2\exp\left(2\pi^2l^2\omega^2\right)\left(\frac{2\sqrt{2}\pi^{3/2}\omega^2}{\sigma_f^2} - \frac{1}{\sqrt{2\pi}l^2\sigma_f^2}\right) \quad (18)$$

and

$$\frac{\partial}{\partial\sigma_f}\mathcal{F}_{rMLE} = -\frac{\sqrt{\frac{2}{\pi}}\hat{y}_i^2\exp\left(2\pi^2l^2\omega^2\right)}{l\sigma_f^3} \quad (19)$$

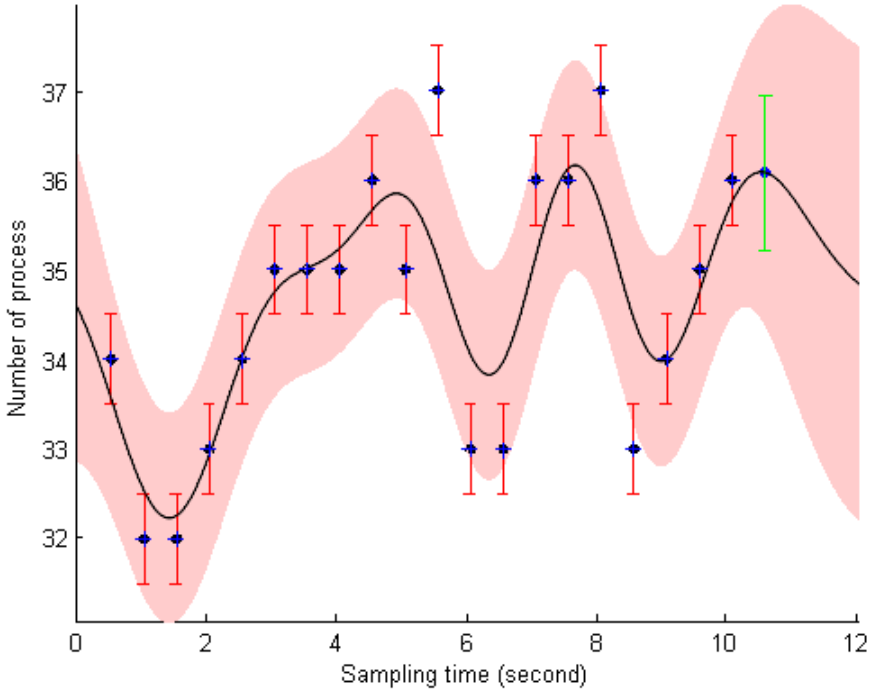


Fig. 2: Mean prediction and error bar of proposed method given 20 training points on stress test

TABLE I: Computation cost of proposed method

	Direct Method	Conjugate Gradient	Proposed Method
Hyper-parameters learning	$O(n^3)$	$O(n^2)$	$O(nlogn)$

After getting the partial derivatives, an updating scheme is issued to update the old hyper-parameters. This scheme is as follows.

$$l^{(k)} \leftarrow l^{(k-1)} + \alpha(k)\frac{\partial}{\partial l^{(k-1)}}\mathcal{F}_{rMLE} \tag{20}$$

$$\sigma_f^{(k)} \leftarrow \sigma_f^{(k-1)} + \alpha(k)\frac{\partial}{\partial \sigma_f^{(k-1)}}\mathcal{F}_{rMLE} \tag{21}$$

in which, $\alpha(k)$ is the decay function with regard to the k^{th} iteration. The decay function is chosen instead of the exact line search or backtracking line search [13] mainly because of the performance issue. For convenience of calculation, a Robbins-Monroe sequence is employed to construct the decay function $\alpha(k) = {}^1/_{(k+1)}$. Along with the above partial derivatives, this decay function ensures the convergence of the optimization procedure.

To govern the number of iteration for the optimization algorithm (in this case, the SGD), an error function is defined based on the Root Mean Square Error (RMSE) method to measure the convergence. Note that the RMSE method is stricter than the frequently-used Mean Square Error (MSE) method. Theoretically, the RMSE threshold is limited to 10^{-11} which produces a solution very close to the real one. Clearly, the core of this optimization procedure is to conduct all the steps in the periodic domain. This means that the optimization can be done with no need of matrix inversion. Additionally, the vector of dual weight $\mathbf{y}_0 \approx \mathbf{\Phi} * \mathbf{y}_\circ$ can also be estimated in the spectral domain. By the end of this hyper-parameter learning phase, the set of hyper-parameters is ready for the Predictor. The comparison of complexity between the proposed method and the others can be found in Table I.

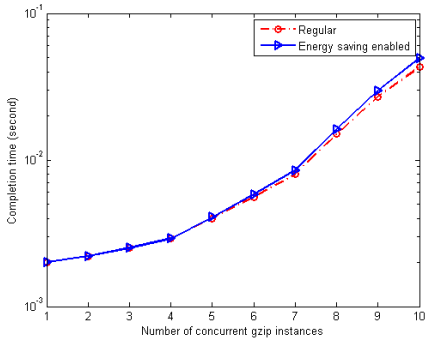
IV. Performance Evaluation

A. Experiments

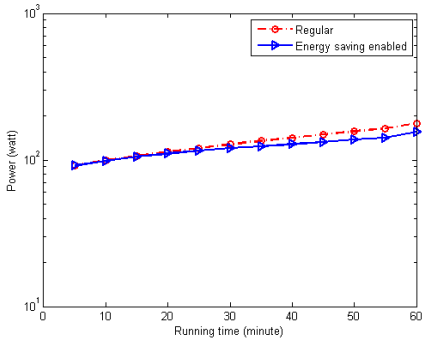
TABLE II: System configuration

	Configuration
Platform	64bit
CPU	Intel®Core™ i7-3770, 3.40GHz
Storage	800GB
Memory	16GB
OS	CentOS 6.5 (final) Kernel: 2.6.32-431.el6.x86_64
Benchmark	*stress*-1.0.4
Power stat	*powerstat*-0.01.30-1
System stat	*sysstat*-9.0.4-27.el6
Gzip-test-data	text file (256 KB)

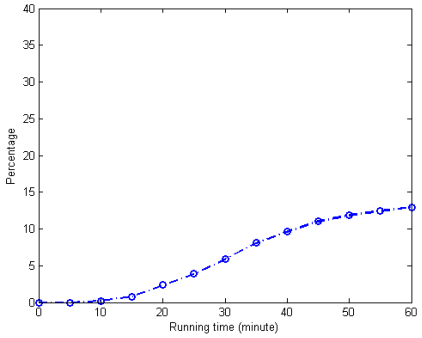
For the performance evaluation, our experiments are aimed



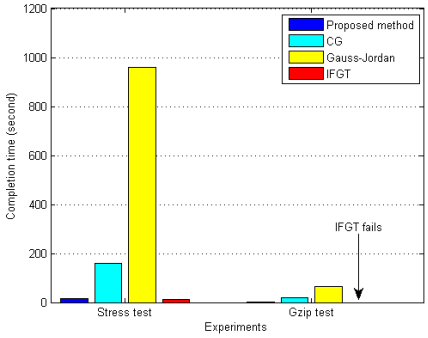
(a) Execution time comparison between two systems (lower is better)



(b) Power consumption over one hour running time (lower is better)



(c) Power saving over one hour running time



(d) Hyper-parameter learning speed evaluation (lower is better)

Fig. 3: Performance evaluation of proposed method

at investigating the performance of the proposed application in terms of energy efficiency and execution time. In the initial experiments, the workload is generated *via* the CPU intensive benchmark for one hour to determine the energy savings. In this test, in order to more easily control the number and the intensiveness of the workload, a benchmark software, namely *stress*-1.0.4, is used to simulate the incoming processes. Otherwise, in the second experiment, ten bunches of ten concurrent jobs (totally one hundred instances of *gzip* command on 256 KB of test data) are pushed into the system to test the execution time. To aggregate the results, the *powerstat* and the *sysstat* software are used to log the power consumption and workload statistics, respectively. All of the information of the benchmarking system is described in Table II.

B. Metrics

The proposed architecture is measured on two levels: the algorithm level and the application level. In the algorithm level, the metrics of interest are the completion time of prediction and the accuracy. In the application level, as previously mentioned, the energy efficiency and execution time are the metrics to be measured. If the application is able to save the energy consumption, as well as to maintain an acceptable execution time, the energy efficiency of the CPU would be significantly improved.

C. Results

Application level - execution time evaluation: in the *gzip* experiment, the system engaging the energy saving application is slightly slower than the regular one, increasing from 2% to 14%. This delay time derives from both predicting the utilization and migrating the processes. In the worst case, despite increasing by 14%, the time gap between two systems is just $6.43*10^{-3}$ seconds which is infinitesimal and acceptable (Figure $3a$).

Application level - energy efficiency evaluation: as seen in the Figure $3b$, both systems begin with the stand-by mode, which costs 91.49 watts to maintain. Both systems had simultaneous stress tests for a duration of 60 minutes. Subsequently, the system with energy saving enabled ends the benchmark test with a consumption of 154.93 watts, in comparison with 177.96 watts for the regular system. Therefore, 23.03 watts are saved (which is equivalent to an energy savings of 12.94%) (Figure $3c$). In processor architecture, an energy reduction of 12,94% is significant.

Algorithm level - completion time: as seen in Figure $3d$, within the same error bound ($\epsilon = 10^{-11}$) and the same training dataset (around 10^3 points), the proposed method approximately takes 17 seconds to finish estimating the hyper-parameters on the stress test, whereas the CG and Gauss-Jordan elimination cost 160 seconds and 960 seconds, respectively. For a different training dataset (100 target points in *gzip*

test), the proposed method needs approximately 1.7 seconds to finish estimating the hyper-parameters, while the CG and Gauss-Jordan elimination cost 20 seconds and 66 seconds, respectively. In particular, for this small test, the original IFGT algorithm tolerates more failure in the computation. This is predominant due to the difficulties inherent in applying the Gaussian-type potential for maximum likelihood estimation.

Algorithm level - accuracy: for the reliability measurement, because the proposed method also partially relies on the IFGT, which defines the precision of $\epsilon = 10^{-11}$ in advance, the accuracy requirements is always satisfied. For an accuracy benchmark of 20 consecutive testing points in the stress experiment, the mean prediction is able to adapt well to the testing data, with 95% confidence maintained by the variance (Figure 2).

V. Conclusion

The proposed method proves the capability in improving the power consumption of the computing node. To do that, the strategy is to predict the utilization of CPU cores, migrate the target processes and stand-by the idle cores to save the energy. To sum up, this method has a major contribution to the field. Based on the knowledge of queuing theory, stochastic process, and optimization, the proposed method reduces the complexity of the hyper-parameter learning phase of the Gaussian process regression technique, from $O(n^3)$ to $O(nlogn)$. With this approach, the prediction on periodic time series event performs faster, more stably, and more reliably. This improvement increases the reaction rate of the prediction method, makes it possible to deal with the large-scale system.

VI. Acknowledgments

This work was partly supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.R0101-15-237,Development of General-Purpose OS and Virtualization Technology to Reduce 30% of Energy for High-density Servers based on Low-power Processors) and was supported by the Industrial Core Technology Development Program (10049079 , Development of mining core technology exploiting personal big data), funded by the Ministry of Trade, Industry and Energy (MOTIE, Korea) and the National Research Foundation of Korea(NRF) grant, funded by the Korea government(MSIP) NRF-2014R1A2A2A01003914.

References

[1] D. Petelin, B. Filipič, and J. Kocijan, "Optimization of gaussian process models with evolutionary algorithms." in *Proceedings of the 10th International Conference on Adaptive and Natural Computing Algorithms - Volume Part I*, ser. ICANNGA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 420–429. [Online]. Available: http://dl.acm.org/citation.cfm?id=1997052.1997098

[2] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data." *CoRR*, vol. abs/1309.6835, 2013.

[3] Y. Shen, A. Y. Ng, and M. Seeger, "Fast gaussian process regression using kd-trees." in *NIPS*, 2005.

[4] C. Yang, R. Duraiswami, and L. S. Davis, "Efficient kernel machines using the improved fast gauss transform." in *NIPS*, 2004.

[5] K. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, "An introduction to kernel-based learning algorithms." *Neural Networks, IEEE Transactions on*, vol. 12, no. 2, pp. 181–201, Mar 2001.

[6] E. G. Tsionas, "Maximum likelihood estimation of stochastic frontier models by the fourier transform." *Journal of Econometrics*, vol. 170, no. 1, pp. 234–248, 2012.

[7] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kgl, "Algorithms for hyper-parameter optimization." in *NIPS*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, Eds., 2011, pp. 2546–2554.

[8] E. Rodner, A. Freytag, P. Bodesheim, and J. Denzler, "Large-scale gaussian process classification with flexible adaptive histogram kernels." in *ECCV (4)*, ser. Lecture Notes in Computer Science, A. W. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds., vol. 7575. Springer, 2012, pp. 85–98.

[9] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, ser. Adaptive Computation And Machine Learning. MIT Press, 2005. [Online]. Available: http://www.gaussianprocess.org/gpml/chapters/

[10] T. T. Cai, T. Liang, and H. H. Zhou, "Law of log determinant of sample covariance matrix and optimal estimation of differential entropy for high-dimensional gaussian distributions." *CoRR*, vol. abs/1309.0482, 2013.

[11] J. de Baar, R. Dwight, and H. Bijl, "Speeding up kriging through fast estimation of the hyperparameters in the frequency-domain." *Computers & Geosciences*, vol. 54, no. 0, pp. 99–106, 2013.

[12] P. Sollich and C. K. I. Williams, "Understanding gaussian process regression using the equivalent kernel." in *Deterministic and Statistical Methods in Machine Learning*, ser. Lecture Notes in Computer Science, J. Winkler, M. Niranjan, and N. D. Lawrence, Eds., vol. 3635. Springer, 2004, pp. 211–228. [Online]. Available: http://dblp.uni-trier.de/db/conf/dsmml/dsmml2004.html

[13] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004. [Online]. Available: http://books.google.co.kr/books?id=mYm0bLd3fcoC

[14] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain." Pittsburgh, PA, USA, Tech. Rep., 1994.

[15] C. Yang, R. Duraiswami, N. Gumerov, and L. Davis, "Improved fast gauss transform and efficient kernel density estimation." in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, Oct 2003, pp. 664–671 vol.1.

[16] T. I. Alecu, S. Voloshynovskiy, and T. Pun, "The gaussian transform." in *EUSIPCO2005, 13th European Signal Processing Conference*, 2005, pp. 4–8.

[17] L. Greengard and J. Strain, "The fast gauss transform." *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 1, pp. 79–94, 1991.

[18] M. Spivak, S. K. Veerapaneni, and L. Greengard, "The fast generalized gauss transform." *SIAM J. Sci. Comput.*, vol. 32, no. 5, pp. 3092–3107, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1137/100790744

[19] R. S. Sampath, H. Sundar, and S. K. Veerapaneni, "Parallel fast gauss transform." in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.39

[20] V. Simoncini and D. B. Szyld, "Theory of inexact krylov subspace methods and applications to scientific computing." *SIAM Journal on Scientific Computing*, vol. 25, no. 2, pp. 454–477, 2003.

[21] V. I. Morariu, B. V. Srinivasan, V. C. Raykar, R. Duraiswami, and L. S. Davis, "Automatic online tuning for fast gaussian summation." in *NIPS*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Curran Associates, Inc., 2008, pp. 1113–1120.

[22] K. Chalupka, C. K. I. Williams, and I. Murray, "A framework for evaluating approximation methods for gaussian process regression." *CoRR*, vol. abs/1205.6326, 2012.