Vol.2016-HPC-156 No.7
2016/9/16

「京」での運用効率改善・電力の効率化に向けた ジョブ解析システムの開発

黒田明義[†1]　井上俊介[†2]　小山謙太郎[†2]　関澤龍一[†3]　南一生[†1]

概要：近年の HPC システムは，システム性能の向上に伴い，様々な課題に直面している．導入コストや運用コストの増大，利用者の増加，利用者層の拡大があげられる．これに起因して，スーパーコンピュータ「京」でも運用効率の向上や電力の効率化が，運用の立場から大きな課題となっている．これらを解決するべく，実際に使用されるアプリケーションの性能解析が必要であるが，現在運用側でそれを把握するのは難しい．理化学研究所計算科学研究機構では，富士通と協力して，「京」の運用効率改善・電力の効率化を目的としたジョブ解析を行うべく，ジョブ情報を自動的に採取し，データベースを構築し，解析するツール群の整備を行った．本報ではその公開にあたり，構築したシステムならびに解析事例を紹介する．

キーワード：ジョブ解析ツール，運用効率改善，省電力，アプリケーション性能解析，スーパーコンピュータ京

Development of Job Analysis System for Power-saving and Operational Improvement on the K computer

AKIYOSHI KURODA[†1]　SHUNSUKE INOUE[†2]　KENTARO KOYAMA[†2]
RYUICHI SEKIZAWA[†3]　KAZUO MINAMI[†1]

Keywords:, Job analysis tool, Operational improvement, Power consumption reduction, Application performance, K computer

1. はじめに

近年の HPC システムは, システム性能の向上による大規模化に伴い，様々な課題に直面している．導入コストや運用コストの増大，利用者の増加，利用者層の拡大などがあげられる．コストの増大により，設置機関ごとにスーパーコンピュータを運用していた形態から，共同利用へと移行が促進し，運用についても一層の効率化が求められるようになった. スーパーコンピュータ「京」(以下「京」と記す) でも計算資源の有効活用やジョブ電力の効率化が，運用の立場から課題となっている．効率的な運用の指標として資源の利用率などが用いられてきたが，これからはジョブを構成するアプリケーションの性能の向上による運用効率改善も視野にいれるべきである．アプリケーション性能向上は，自身の利用時間の短縮のみならず，他の利用者の待ち時間の削減効果をもたらし，資源の有効利用につながる．

一般にアプリケーションの高速化は，利用者側にて自発的に情報採取し，行う必要があった．運用側でこれを行うためには，使用されているアプリケーションの理解が必要である．しかし共同利用による利用者層の拡大や，ブラックボックスとしてアプリケーションを使用するユーザの増加などにより，運用側でアプリケーションの性能を掌握するのは難しくなってきている．

理化学研究所計算科学研究機構では，富士通と協力して，「京」の運用効率改善・ジョブ電力の効率化を目的として，ジョブの性能情報を運用システムで自動的に採取し，それらを蓄積するデータベースを構築し，解析するツール群を整備した．本報ではその公開にあたり，構築したシステムを紹介する．

第 2 章では，本システムを導入した「京」の特徴について述べ，運用効率改善とアプリケーション性能の関係ついて説明する．第 3 章では，採取した性能について採取方法など詳細に説明する．第 4 章では，採取したデータを蓄積するデータベースについて説明する．第 5 章では，構築した性能データベースを解析するツールについて説明し，実際の解析例を紹介する．第 6 章では収集したデータをユーザにフィードバックするために開発した公開ツールについて紹介する．第 7 章では，現状の問題を議論しまとめる．

2. 自動性能採取システム

2.1 スーパーコンピュータ「京」

自動性能採取システム開発に使用した「京」について紹介する．CPU は富士通社製の SPARC64^{TM} VIIIfx [1][2][3]であり，CPU と周辺 LSI から構成される計算ノードは，浮動小数点演算のピーク性能が 128[GFLOPS]，メモリ量が 16[GB]，メモリの理論バンド幅が 64[GB/s]である．ユーザが利用できる総計算資源は，82,944 ノード，ピーク性能 10.62[PFLOPS]，メモリ量 1.26[PB]である．

ノードはインターコネクト用 LSI（ICC: Inter-Connect Controller）を通じて，Tofu インターコネクトと呼ばれる 6

†1 理化学研究所　RIKEN
†2 株式会社富士通システムズ・イースト　Fujitsu Systems East Limited.
†3 富士通株式会社　FUJITSU, LTD.

次元メッシュ/トーラスネットワークで結合される[4][5]. 各リンクの理論バンド幅は 5[GB/s]（双方向）で，4 方向の同時通信が可能である.

ファイルシステムは，グローバル及びローカルの 2 階層で構成されている．ジョブが使用する計算ノード直下のローカルファイルシステムは 11[PB]であり，性能はラックあたり最大で約 3[GB/s]で，ノードあたり約 30[MB/s]である．Tofu ネットワークの一部を使用する.

2.2 運用効率とアプリケーション性能

システムの運用効率を改善するためには，ジョブの隙間を減らし利用率を高める方法がある．しかし，「京」ではジョブの利用率は常時 80%を越えており，直接網ネットワークで隙間なくジョブを埋め込むには限界がある．運用効率を改善する他の手段として，ジョブが早く終了するよう，使用するノード時間積そのものを小さくする方法がある．このためにはジョブの性能を高める必要があり，流れているジョブの性能情報を透過的に採取することが必要とされる.

アプリケーションの性能は演算だけでは評価できない．我々は「京」で利用されるいくつかのアプリケーションを評価した結果，様々な評価指標のうち演算性能，メモリスループットの 2 点を採取し，評価できれば，一定の性能分析が可能であるとの結論に達した．例えば，「京」で測定された LINPACK 性能は 10.510[PFLOPS]であったが[6]，この時の演算効率は 93.17%であり，システムの演算性能を効果的に利用していると言える．HPCG の性能は，0.554[PFLOPS][7]，演算効率は 5.22%であり，演算性能を効果的に利用しているとはいえない．しかしメモリスループットは，区間毎に 44.3[GB/s]～47.8[GB/s]であった[8]．ストリームベンチマークの結果によると，メモリバンド幅 64[GB/s]のうち，達成可能なメモリスループットの上限は 46[GB/s]程度とされ，HPCG はメモリ性能を効果的に利用しているといえる．これらはベンチマークプログラムであるが，どちらもシステム性能を効果的に利用しているアプリケーションといえる．実際のジョブの多くは，これら 2 つの性能指標が供に改善が可能であり，これらを高めることは，システムへの運用効率改善へつながる.

性能を押し下げる要因として，インバランスが考えられる．インバランスはスレッド間の演算のばらつきや，通信，I/O などの要因で発生する．このことから，効率改善に資するジョブ情報として，①性能情報，②インバランスの 2 つに着目し，自動採取を行うことにした.

運用にとって，ジョブ電力の効率化は，運用効率改善と並んで大きな課題の 1 つである．ジョブが消費する電力について，性能と消費電力の関係について調べられている[9]. ここで提案されている方法では，演算に関する性能や，記憶領域に対するアクセス性能を指標として，以下のように評価している.

$$
\begin{aligned}
&Power[MW/sys] \\
&=8.8128 + (1.3659 * FLOPS[\%] + 0.2429 * MIPS[\%] \\
&+ 4.3906 * Memory[\%] + 0.0857 * L2[\%] \\
&+ 2.3299 * L1[\%]) / 100 \qquad (1)
\end{aligned}
$$

この式からアプリケーションが消費する電力は，性能に依存しない待機電力と，性能に比例して増加する比例電力の 2 つに大別できる．アプリケーション性能向上に伴い，比例電力の分，消費する電力は増える．しかし実行時間が短縮できるため，時間積分としての比例電力量は変わらないが，待機電力は実行時間が短縮した分，アプリケーションが消費する電力量の節約効果が期待できる（図 1）.

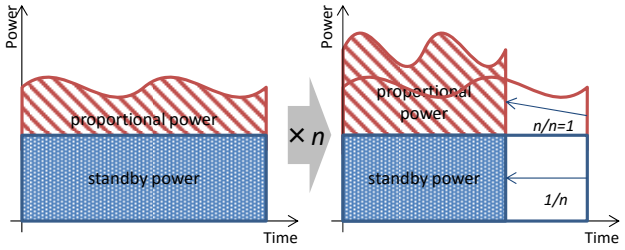

図 1　アプリケーションの消費電力と性能の関係.

Figure 1　Relation between power consumption and application performance.

一般にスーパーコンピュータでは，様々な情報収集がなされている．施設では電力や温度の情報，運用ではシステムの運用状況，課金に関係する情報などユーザが実施している計算に関する情報を収集している．しかしこれらの情報は，目的ごとに管理されていることが多く，その関連性を解析するのは難しい．運用効率の改善に資する情報として，ジョブの性能情報は重要である．実際に演算に関する性能を採取できるシステムは多く，「京」でも，一部の性能情報を自動的に採取し，マンスリーレポートとしてユーザにフィードバックしている．しかしこれらの情報は，課金管理を目的としており，運用改善を目的としていない形式であった。

「京」では，ハードウェアカウンタを用いて演算性能を採取しているが，使用するカウンタのパラメータを変更することで，運用改善に必要となる様々な性能情報を取得することが可能であることが分かった．また，性能情報は，ハードウェアカウンタ情報だけでなく，ユーザに標準で提供されているジョブ統計情報や OS の持つカウンタ情報，ノードが管理している情報などがあり，それらの多くは性能解析に利用可能である.

3. 解析可能なジョブ情報

2.2 節で説明したように，システムの運用改善にはアプリケーションの情報が不可欠である．運用改善に必要とさ

れるアプリケーションの情報として，①性能情報，②インバランスに関する情報，③アプリケーションに附随するその他の情報があげられる．本章では今回採取した情報について，その採取方法について個別に説明する．

3.1 性能情報

(1) 浮動小数点演算性能

浮動小数点演算性能は，ハードウェアカウンタ情報を用いて採取可能である．これはユーザがアプリケーションの性能を測定するときに用いる手法を用いる方法である．使用するハードウェアカウンタは，演算量に関する 4 種類（*floating_instructions, SIMD_floating_instructions, SIMD_fma_instructions, fma_instructions*）であり，その他にジョブ統計情報の実行時間（*elapse*）と使用ノード数（*use_node_num*）を用いて，以下の数式で算出可能である．

$$FLOPS[\%] = (floating\_instructions + SIMD\_floating\_instructions * 2 + SIMD\_fma\_instructions * 4 + fma\_instructions * 2) / use\_node\_num / elapse / 128G * 100 \quad (2)$$

(2) メモリスループット

アプリケーションの中には，HPCG のように演算は少なく，記憶領域に頻繁にアクセスするものが多く存在する．流体や構造のステンシル計算はその典型的であり，それらのアプリケーションを演算性能だけで評価するのは意味がない．記憶領域に関する性能は，演算性能同様ハードウェアカウンタを用いて採取可能である．記憶領域は，L1 キャッシュ, L2 キャッシュ, メインメモリの階層構造を持つが，利用可能なハードウェアカウンタ数の制限から，全てを一度に採取することはできない．流体や構造解析といったアプリケーションはメモリインテンシブであることが多く，キャッシュを有効活用しているアプリケーションは，演算器を有効に活用していることが多い[10]ため，メインメモリの性能をのみを採取した．解析に使用するカウンタは，サイクル数（*cycle_counts*）とメモリアクセスに関するカウンタ 2 種類（*cpu_mem_read_count, cpu_mem_write_count*）であり，

$$Memory[GB/s] = (cpu\_mem\_read\_count + cpu\_mem\_write\_count) * 128 / (cycle\_counts / 2G) / 1G \quad (3)$$

にて算出可能である．ここで用いられる定数 128 はキャッシュラインサイズ，2G はサイクル数から時間を換算する CPU のクロック数であり, 1G は単位の換算ために用いた．

3.2 インバランス情報

性能を押し下げる要因として，インバランスがあげられる. 評価には, CPU が待機しているサイクル数（*sleep_cycle*）をあらわすハードウェアカウンタを用いた. この時間には，通信中のマスタースレッド以外待ち状態や，スレッドインバランスが含まれる．これらの切り分けは難いため，評価は通信とスレッドインバランスを合わせた量で行った．インバランスには，*sleep_cycle* の他に，CPU の休止状態も考えられる．図 2 を見ると working の領域は CPU が可動している時間で, waiting の領域は sleep している時間である．その他 I/O 時間の待ちや，Flat MPI などで thread そのものを立ち上げていない時なども考えられ, 休止状態として not running の領域で表現される. これらは全てインバランスに含めるべきと考え, waiting と not running の領域を合わせた時間で評価を行った．使用するハードウェアカウンタは，サイクル数（*cycle_counts*）とスリープサイクル数（*sleep_cycle*）で，その他にジョブ統計情報の実効時間（*elapse*）と使用ノード数（*use_node_num*）を用いて，

$$sleep\_time[s] = elapse - (cycle\_counts - sleep\_cycle) / 2G / (use\_node\_num * 8) \quad (4)$$

で算出され，8 つ全てのハードウェアカウンタを利用したことになる. 使用コア数の計算に *use_node_num*8* を使用しているが，ジョブ統計情報の *alloc_core_total* を用いると，使用しない割り当てノードも使用コア数に含まれる．また *use_core_total* を用いると OS などが使用したコア数を含む実際の使用したコア数となるため注意が必要である．

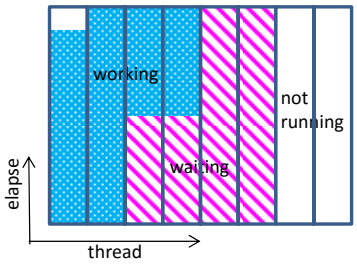

図 2　sleep time の概念図．waiting は，通信やロードインバランスの待ち時間で，not running は，CPU が休止している時間であり，その合計を sleep time として評価した．

Figure 2　The concept of sleep time. We took the sum of the waiting regions and not running regions to be the sleep time.

(1) 通信情報

インバランスの要因の一つとして通信に関する情報は，ハードウェアカウンタを用いて採取することは難しい．かわりに OS の機能である Sar 情報を用いて，通信量の評価が可能である.「京」では，ICC が持つカウンタを使用し，Tofu ネットワーク向けに Sar 情報を拡張しており，採取可能である. 使用するカウンタは, *tnuser_in_num, tnuser_in_sz, tnuser_out_num, tnuser_out_sz* であり，それぞれ，ノードが入出力した通信回数とデータ量を意味する．

Sar 情報を採取するためには，採取開始と終了のタイミングでシグナルを送る必要がある．現状の「京」でのシステム改変を最小限にするためには，ジョブ終了時にそのシグナルを送ることは可能であるが，開始時に送るのは難しい．そこで，前回のジョブ終了時～現在のジョブ終了時のカウンタ情報を使用することで，通信量を評価した．

また Sar は，ジョブ単位ではなく計算ノード毎に持つ情報である．これらの情報をジョブ単位に集計し，採取するには膨大な時間が必要であることが分かった．ある 1 日の全ジョブの情報は，50 項目のデータからなるファイルが 765,347 存在し，データベースへの登録に 19 時間必要であった．個々のノードの通信情報は，アプリケーション性能評価には必要ないため，最大通信量，平均通信量，最小通信量のみを集計し，更にファイルをデータベース登録前に集約することで処理時間の短縮を図った．

集団通信では，Sar 情報に中継した通信も含まれる．また Sar 情報は，通信量は取得できるが，通信時間に換算できない．このため冗長に通信しているアプリケーションや通信にインバランスのあるアプリケーションを抽出することは可能である．しかし通信サイズやアルゴリズムによって達成可能な通信性能が異なるため，通信時間や全体時間に与える影響が評価できない．

(2) I/O 情報

インバランスの要因の 1 つとして I/O に関する情報は，通信時間と同様ハードウェアカウンタから評価が難しい．しかし I/O 量は，ユーザに提供されているジョブ統計情報から取得可能である．しかし通信同様，I/O 量を性能や時間に換算することは難しい．I/O の場合，集団通信のような多様なアルゴリズムが存在するわけではないため，性能に影響する主要な要因は，データ移動経路の競合やインバランスである．I/O インバランスでは，1 つのファイルに同時にアクセスすることでファイルロックがかかり性能低下が発生する．また一部のノードのみにアクセスがあると帯域不足で性能劣化が発生し得る．これらは，システムの安定稼働にも影響し，問題であるため評価が必要とされた．

インバランスの評価には，ジョブ統計情報の元となるノードごとに格納されている情報を採取することで可能である．通信情報と同様に，ノード情報は膨大であるため，インバランスの評価のために，最大，平均，最小の I/O 量を採取した．

3.3 アプリケーションに附随するその他の情報

(1) 分野情報

一般的にアプリケーションは，アルゴリズムによって達成できる性能が異なるため，性能の絶対値は意味をなさない．このため，アプリケーションに附随する情報として，アルゴリズム情報の採取方法を考えた．

しかし，アルゴリズムに関する情報は，どこのデータベースにも存在せず採取が難しい．代替するものとして，登録機関の協力を得て，課題登録時の分野情報を採取した．課題登録時の分野としては，以下の 8 つがある．「バイオ・ライフ」，「物質・材料・化学」，「環境・防災・減災」，「工学・ものづくり」，「物理・素粒子・宇宙」，「数理科学」，「原子力・核融合」，「その他」である．これに京運用や高度化枠の「その他」の一部を組み込んだ「情報・計算機科学」を含めた 9 つによる分類を行った．

(2) 消費電力

「京」では受電システムやラック毎に設置された電力測定装置が，また一部のラックに精密な電力測定装置が設置され消費電量を測定している．それらの，測定値はそれぞれにデータベースとして蓄積されているが，これらの情報は，ジョブ実行単位である Tofu 単位（12 ノード）とくらべて粒度が荒く，ジョブが消費する電力の評価としては不適応である．ジョブの消費電力については，CPU の冷却温度やシステムボードの排気温度につけられている温度情報から，推算する試み[11]や，性能情報からの推定の試み[9]がなされている．ここでは，性能パラメータとして常時採取可能であり，消費電力への影響が大きいとされる，浮動小数点演算性能とメモリスループットとの関係を再評価し，その値を採取することとした．評価に用いた関係式は，以下の通りである．

$$Power[MW/sys] = 8.8128 + (1.7895 * FLOPS[\%] + 4.9921 * Memory[\%]) / 100 \quad (5)$$

4. ジョブ情報データベース

第 3 章で説明したジョブに関する様々な情報は，各種サーバに分散管理されており，運用効率改善に向けた解析が難しい．主な情報は，ハードウェアカウンタ情報，ジョブ統計情報，利用ノード情報である．

我々は，運用効率改善やユーザへの情報発信に向けて，収集した様々な情報を一元管理し，蓄積するシステムを構築した．構築したデータベースの概念図は，図 3 の通りである．各サーバに分散管理されている情報は，毎日定められた時刻に自動的に収集される．第 3 章で説明した性能や分野，消費電力は，収集された情報から二次的に集計し，データベースを毎日更新した．

性能を採取し，データベースを構築するにあたり，ユーザへの影響はないと考える．これらの性能情報は，ハードウェアやシステムソフトウェアが所有するカウンタ情報を利用しているためであり，更にジョブの終了後に採取し集計されるためである．

いくつかのジョブでは，性能を自動的に採取することができない．小さなジョブをまとめて投入するために導入されたバルクジョブでのジョブ情報を管理する親ジョブや，

ステージインで失敗したジョブなどがあげられる．またユーザが性能取得しているジョブでは，ハードウェアカウンタが不足し，システムで情報を採取できない．性能を蓄積する二次集計では，これらのジョブを除き，正常終了もしくは，制限時間オーバーしたジョブを集計した．

このシステムで得られた情報を公開することで，ユーザは能動的にプロファイルを採取しなくとも、性能の概要を把握することが出来る．但し，現段階で上記で述べた情報のうち I/O 情報の登録だけは自動化されていない．

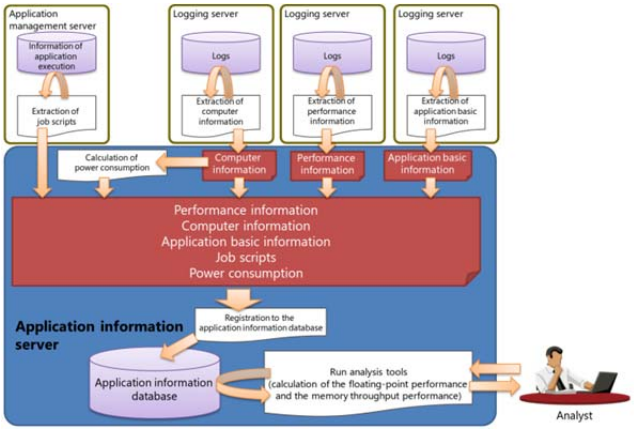

図 3　ジョブ情報データベース概念図.

Figure 3　Concept of diagram job information database.

5. 解析ツール並びに解析例

構築されたデータベースは，1 年で 1GB 程であり，今後新たな情報を取り込むことで，更なる増加が見込まれる．これらの大きなデータを統計処理するのは難しさを伴うため, 解析に向けたツール群を整備した. 用意したツールは，分野やグループ，ジョブ毎に集計するツールなどである．

ジョブの解析を行うと，大規模かつ長時間ジョブは全体の計算資源に占める割合が高いことが分かった．性能の平均処理など統計的な評価をする際には，資源を多く占有するジョブの情報が必要である．このため評価で用いる平均処理には，ジョブ数ではなく，ノード時間積の荷重平均を用いた．解析例をいくつか示す．

表 1 は，課題申請時の分野を用いて，その性能の一覧を集計したものである．浮動小数点演算性能をあらわす flops%を見ると，Material と Physics の分野の性能が比較的高いことがわかる．これは Material では，量子計算での波動関数の処理などで演算密度の高い行列積計算（Level 3 BLAS）を活用することが多いことや，Material，Physics 供に，多体粒子問題を扱うことによるものが多いためと想定される. また Environment と Engineering を見ると, memory%の性能が高い．これらの分野では，構造や流体を扱うことが多く，データアクセスの多いステンシル計算が多いことを反映しているものと想定される．

但し，ここで得られたジョブの演算やメモリスループットの平均性能は，一般的なアプリケーション高速化で評価される性能よりも低い．今回の評価はジョブ単位で行っているため，ジョブ立ち上げのレイテンシや，アプリ本体が動くまでの I/O や初期化などが含まれることが考えられる．しかし，それ以上に性能を押し下げる原因となる I/O のインバランスや Sleep Time の割合も高いことが分かった．

表 1　分野毎の性能一覧

Table 1　Performance table of each field

field	count	node_time	flops%	memory%	sleep%	MWh
Engineering	89,469	92,852,391	2.53%	20.66%	42.44%	11,071
Environment	140,948	89,815,154	4.24%	25.02%	52.53%	10,977
Life	54,480	71,470,748	3.16%	7.10%	31.40%	7,948
Material	73,980	113,000,194	12.18%	13.87%	35.94%	13,246
Mathematics	2,344	7,954,788	2.14%	7.23%	72.74%	883
Nuclear	5,200	4,154,692	8.97%	50.95%	24.68%	577
Physics	40,701	84,041,418	13.69%	20.53%	29.04%	10,216
System	122,449	13,480,385	25.34%	7.65%	56.18%	1,568
None	1,432	1,662,375	11.79%	22.09%	34.58%	203
Sum/Ave	531,003	478,432,143	7.91%	17.50%	39.51%	56,690

図 4 は, Flat MPI 並列と Hybrid MPI 並列の比率を示したグラフである．課題登録情報から産業利用かそれ以外の学術利用かで評価を行った．学術利用での全ジョブに対する Flat MPI 並列ジョブの資源占有率で占める割合は 19.6%であった．それに対して，産業利用では，56.6%と約 3 倍の利用率であることが分かる．これは学術利用では，研究テーマに即したアプリケーションを，課題ごとに独自に開発していることが多いのに比べ，産業利用では，成果のスループットが重要視とされるためオープンソースソフトウェアを活用しているものが多いことによるものと想定される．

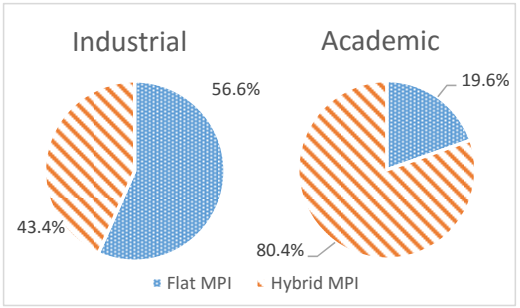

図 4　利用形態による Flat MPI の割合

Figure 4　The proportion of flat MPI through purpose of use

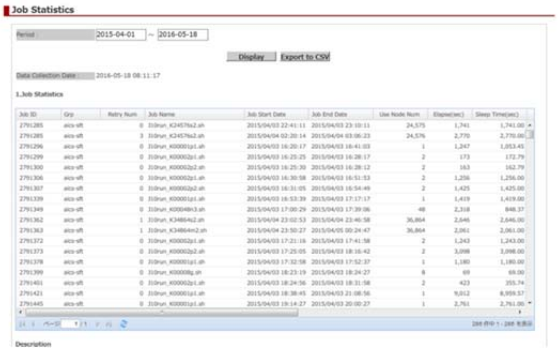

図 5　ユーザのジョブ情報の表示例

Figure 5　Display example of the user job information

6. 公開ツール

ここで集計した情報は，今までのマンスリーレポートを拡張し，ユーザ向けにポータルで公開することを考えている．公開予定の情報は，比較的精度が高い，資源使用量情報，演算性能，メモリスループット，sleep time などで，ユーザは運用で採取できたジョブ全てについて，一覧で参照することが可能である（図 5）．

またこれらの情報は可視化されており，各ユーザのジョブ情報の分布（点）の他に，グループ平均（実線），分野平均（波線）が表示されており（図 6），ユーザは自分のジョブ性能との比較が可能である．

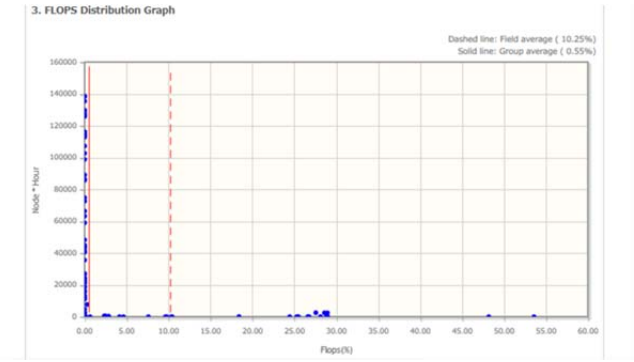

図 6　演算性能の分布図表示例

Figure 6　Distribution diagram example of an operational performance

ユーザ側で独自に性能を採取している場合は，ハードウェアカウンタを占有しているため，運用で透過的に性能を採取できない制限がある．しかし，一般的に多くの資源を使用するとされるプロダクションランでは，ユーザが意識して性能を採取することは稀であると想定され，多くのプロダクションランの性能を網羅的に知ることが可能である．

7. 今後の展開

現在データベースで蓄積が自動化されていない情報がある．I/O インバランス情報，登録情報などがこれにあたり，自動化を手がけている．また新たな試みとして，ロードモジュール情報の蓄積を試行している．使用した言語，ライブラリ情報などである．消費電力情報については，性能情報から算出した予測電力が蓄積されている．しかし，ラックやシステムに紐付けられた電力情報や，温度情報から算出された推定電力なども蓄積も可能である．3.3 節で述べたように，電力測定装置の粒度は，システムやラック単位であり，ジョブの粒度に比べて粗いため，必ずしもジョブ本体の消費電力をあらわさない．しかし，ジョブが含まれるラックの電力合計や，ジョブが占有するラックの電力合計などの情報は，ジョブの消費電力もしくは，ジョブがシステム消費電力へ与えた影響度を知ることが可能であり，温度情報から推算した消費電力や，性能情報から推算した消費電力との比較や推算式へのフィードバックによる精度向上も可能であろう．

これらの情報は，各サーバによって管理形態が異なり，採取の際にデータ欠損などの事象が発生している．これらの欠損や異常を解析し，修復する機能についても検討している．

今後これらの情報を解析することで，実運用におけるシステムの運用効率改善やジョブ電力の効率化に役立てることを目指す．またユーザの協力を得ながら，特に改善効果の高いアプリケーションについて詳細に調査を行い，運用効率改善に実際どの程度効果があるか評価する予定である．これらの試みは，より性能が向上し，複雑化する将来の HPC において，一層重要になると考える．

謝辞　本報告に際し，庄司文由氏，宇野篤也氏，井上文雄氏，寺井優晃氏，山本啓二氏を初めとする理化学研究所計算科学研究機構運用技術部門に有用な意見を頂いた．これらの諸氏に感謝しますと供に，松井秀司氏を初めとする富士通株式会社 SE の諸氏に感謝します．本論文の結果は，理化学研究所計算科学研究機構が保有するスーパーコンピュータ「京」によるものです．

参考文献

[1] Maruyama, T.: SPARC64 VIIIfx: Fujitsu's New Generation Octo-core Processor for Peta Scale Computing., Hot Chips 21 (2009).
[2] Maruyama, T.: SPARC64 VIIIFX: A New-Generation Octocore Processor for Petascale Computing, IEEE micro, Vol.30, No.2, pp30-40 (2010).
[3] SPARC64VIIIfx Extensions, Fujitsu Ltd., architecture manual (2008).
[4] Ajima, Y., Sumimoto, S. and Shimizu, T.: Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers., IEEE Computer, pp.36-40 (2009).
[5] Toyoshima, T., ICC: An interconnect controller for the Tofu interconnect architecture., Hot Chips 22 (2010).
[6] TOP500 Supercomputer Sites https://www.top500.org/
[7] HPCG http://www.hpcg-benchmark.org/
[8] Kumahata, K.; Minami, K. and Maruyama, N.: “High-performance conjugate gradient performance improvement on the K computer”, International Journal of High Performance Computing Applications, Vol.30, No.1, pp.55-70 (2016).
[9] 黒田 明義, 北澤 好人, 塚本 俊之, 小山 謙太郎, 井上 晃, 南 一生: スーパーコンピュータ「京」を用いたアプリケーション性能特性と使用電力の相関解析, 情報処理学会論文誌 コンピューティングシステム (ACS), Vol8, No.4, pp.1-12 (2015-11).
[10] 南一生: 京速コンピュータ「京」におけるアプリケーション高性能化, 電子情報通信学会誌, Vol.95, No.2, pp.125-130 (2012).
[11] 宇野篤也，肥田元，井上文雄，池田直樹，塚本俊之，末安史親，松下聡，庄司文由：消費電力を考慮した「京」の運用方法の検討，情報処理学会論文誌 ACS)，Vol.8，No.4，pp.13-25 (2015).