Dror Feitelson
Eitan Frachtenberg
Larry Rudolph
Uwe Schwiegelshohn (Eds.)

LNCS 3834

Job Scheduling Strategies for Parallel Processing

11th International Workshop, JSSPP 2005
Cambridge, MA, USA, June 2005
Revised Selected Papers

Springer

Dror Feitelson Eitan Frachtenberg
Larry Rudolph Uwe Schwiegelshohn (Eds.)

Job Scheduling Strategies for Parallel Processing

11th International Workshop, JSSPP 2005
Cambridge, MA, USA, June 19, 2005
Revised Selected Papers

Springer

Volume Editors

Dror Feitelson
The Hebrew University, School of Computer Science and Engineering
91904 Jerusalem, Israel
E-mail: feit@cs.huji.ac.il

Eitan Frachtenberg
Los Alamos National Laboratory, Computer and Computational Sciences Division
Los Alamos, NM 87545, USA
E-mail: etcs@cs.huji.ac.il

Larry Rudolph
Massachusetts Institute of Technology, CSAIL
32 Vassar Street, Cambridge, MA 02139, USA
E-mail: rudolph@csail.mit.edu

Uwe Schwiegelshohn
University of Dortmund, Robotics Research Institute (IRF-IT)
44221 Dortmund, Germany
E-mail: uwe.schwiegelshohn@udo.edu

Library of Congress Control Number: 2005937592

CR Subject Classification (1998): D.4, D.1.3, F.2.2, C.1.2, B.2.1, B.6, F.1.2

ISSN 0302-9743
ISBN-10 3-540-31024-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-31024-2 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2005
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11605300 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3834

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison
Lancaster University, UK

Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler
University of Surrey, Guildford, UK

Jon M. Kleinberg
Cornell University, Ithaca, NY, USA

Friedemann Mattern
ETH Zurich, Switzerland

John C. Mitchell
Stanford University, CA, USA

Moni Naor
Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz
University of Bern, Switzerland

C. Pandu Rangan
Indian Institute of Technology, Madras, India

Bernhard Steffen
University of Dortmund, Germany

Madhu Sudan
Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos
New York University, NY, USA

Doug Tygar
University of California, Berkeley, CA, USA

Moshe Y. Vardi
Rice University, Houston, TX, USA

Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Lecture Notes in Computer Science

For information about Vols. 1–3748

please contact your bookseller or Springer

Vol. 3860: D. Pointcheval (Ed.), Topics in Cryptology – CT-RSA 2006. XI, 365 pages. 2006.

Vol. 3850: R. Freund, G. Păun, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing. IX, 371 pages. 2006.

Vol. 3838: A. Middeldorp, V. van Oostrom, F. van Raamsdonk, R. de Vrijer (Eds.), Processes, Terms and Cycles: Steps on the Road to Infinity. XVIII, 639 pages. 2005.

Vol. 3837: K. Cho, P. Jacquet (Eds.), Technologies for Advanced Heterogeneous Networks. IX, 307 pages. 2005.

Vol. 3835: G. Sutcliffe, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. XIV, 744 pages. 2005. (Sublibrary LNAI).

Vol. 3834: D. Feitelson, E. Frachtenberg, L. Rudolph, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing. VIII, 283 pages. 2005.

Vol. 3833: K.-J. Li, C. Vangenot (Eds.), Web and Wireless Geographical Information Systems. XI, 309 pages. 2005.

Vol. 3829: P. Pettersson, W. Yi (Eds.), Formal Modeling and Analysis of Timed Systems. IX, 305 pages. 2005.

Vol. 3828: X. Deng, Y. Ye (Eds.), Internet and Network Economics. XVII, 1106 pages. 2005.

Vol. 3827: X. Deng, D. Du (Eds.), Algorithms and Computation. XX, 1190 pages. 2005.

Vol. 3826: B. Benatallah, F. Casati, P. Traverso (Eds.), Service-Oriented Computing - ICSOC 2005. XVIII, 597 pages. 2005.

Vol. 3824: L.T. Yang, M. Amamiya, Z. Liu, M. Guo, F.J. Rammig (Eds.), Embedded and Ubiquitous Computing – EUC 2005. XXIII, 1204 pages. 2005.

Vol. 3823: T. Enokido, L. Yan, B. Xiao, D. Kim, Y. Dai, L.T. Yang (Eds.), Embedded and Ubiquitous Computing – EUC 2005 Workshops. XXXII, 1317 pages. 2005.

Vol. 3822: D. Feng, D. Lin, M. Yung (Eds.), Information Security and Cryptology. XII, 420 pages. 2005.

Vol. 3821: R. Ramanujam, S. Sen (Eds.), FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science. XIV, 566 pages. 2005.

Vol. 3820: L.T. Yang, X. Zhou, W. Zhao, Z. Wu, Y. Zhu, M. Lin (Eds.), Embedded Software and Systems. XXVIII, 779 pages. 2005.

Vol. 3819: P. Van Hentenryck (Ed.), Practical Aspects of Declarative Languages. X, 231 pages. 2006.

Vol. 3818: S. Grumbach, L. Sui, V. Vianu (Eds.), Advances in Computer Science – ASIAN 2005. XIII, 294 pages. 2005.

Vol. 3816: G. Chakraborty (Ed.), Distributed Computing and Internet Technology. XXI, 606 pages. 2005.

Vol. 3815: E.A. Fox, E.J. Neuhold, P. Premsmit, V. Wuwongse (Eds.), Digital Libraries: Implementing Strategies and Sharing Experiences. XVII, 529 pages. 2005.

Vol. 3814: M. Maybury, O. Stock, W. Wahlster (Eds.), Intelligent Technologies for Interactive Entertainment. XV, 342 pages. 2005. (Sublibrary LNAI).

Vol. 3813: R. Molva, G. Tsudik, D. Westhoff (Eds.), Security and Privacy in Ad-hoc and Sensor Networks. VIII, 219 pages. 2005.

Vol. 3810: Y.G. Desmedt, H. Wang, Y. Mu, Y. Li (Eds.), Cryptology and Network Security. XI, 349 pages. 2005.

Vol. 3809: S. Zhang, R. Jarvis (Eds.), AI 2005: Advances in Artificial Intelligence. XXVII, 1344 pages. 2005. (Sublibrary LNAI).

Vol. 3808: C. Bento, A. Cardoso, G. Dias (Eds.), Progress in Artificial Intelligence. XVIII, 704 pages. 2005. (Sublibrary LNAI).

Vol. 3807: M. Dean, Y. Guo, W. Jun, R. Kaschek, S. Krishnaswamy, Z. Pan, Q.Z. Sheng (Eds.), Web Information Systems Engineering – WISE 2005 Workshops. XV, 275 pages. 2005.

Vol. 3806: A.H. H. Ngu, M. Kitsuregawa, E.J. Neuhold, J.-Y. Chung, Q.Z. Sheng (Eds.), Web Information Systems Engineering – WISE 2005. XXI, 771 pages. 2005.

Vol. 3805: G. Subsol (Ed.), Virtual Storytelling. XII, 289 pages. 2005.

Vol. 3804: G. Bebis, R. Boyle, D. Koracin, B. Parvin (Eds.), Advances in Visual Computing. XX, 755 pages. 2005.

Vol. 3803: S. Jajodia, C. Mazumdar (Eds.), Information Systems Security. XI, 342 pages. 2005.

Vol. 3802: Y. Hao, J. Liu, Y.-P. Wang, Y.-m. Cheung, H. Yin, L. Jiao, J. Ma, Y.-C. Jiao (Eds.), Computational Intelligence and Security, Part II. XLII, 1166 pages. 2005. (Sublibrary LNAI).

Vol. 3801: Y. Hao, J. Liu, Y.-P. Wang, Y.-m. Cheung, H. Yin, L. Jiao, J. Ma, Y.-C. Jiao (Eds.), Computational Intelligence and Security, Part I. XLI, 1122 pages. 2005. (Sublibrary LNAI).

Vol. 3799: M. A. Rodríguez, I.F. Cruz, S. Levashkin, M.J. Egenhofer (Eds.), GeoSpatial Semantics. X, 259 pages. 2005.

Vol. 3798: A. Dearle, S. Eisenbach (Eds.), Component Deployment. X, 197 pages. 2005.

Vol. 3797: S. Maitra, C. E. V. Madhavan, R. Venkatesan (Eds.), Progress in Cryptology - INDOCRYPT 2005. XIV, 417 pages. 2005.

Vol. 3796: N.P. Smart (Ed.), Cryptography and Coding. XI, 461 pages. 2005.

此为试读,需要完整PDF请访问: www.ertongbook.com

Vol. 3795: H. Zhuge, G.C. Fox (Eds.), Grid and Cooperative Computing - GCC 2005. XXI, 1203 pages. 2005.

Vol. 3794: X. Jia, J. Wu, Y. He (Eds.), Mobile Ad-hoc and Sensor Networks. XX, 1136 pages. 2005.

Vol. 3793: T. Conte, N. Navarro, W.-m.W. Hwu, M. Valero, T. Ungerer (Eds.), High Performance Embedded Architectures and Compilers. XIII, 317 pages. 2005.

Vol. 3792: I. Richardson, P. Abrahamsson, R. Messnarz (Eds.), Software Process Improvement. VIII, 215 pages. 2005.

Vol. 3791: A. Adi, S. Stoutenburg, S. Tabet (Eds.), Rules and Rule Markup Languages for the Semantic Web. X, 225 pages. 2005.

Vol. 3790: G. Alonso (Ed.), Middleware 2005. XIII, 443 pages. 2005.

Vol. 3789: A. Gelbukh, Á. de Albornoz, H. Terashima-Marín (Eds.), MICAI 2005: Advances in Artificial Intelligence. XXVI, 1198 pages. 2005. (Sublibrary LNAI).

Vol. 3788: B. Roy (Ed.), Advances in Cryptology - ASIACRYPT 2005. XIV, 703 pages. 2005.

Vol. 3785: K.-K. Lau, R. Banach (Eds.), Formal Methods and Software Engineering. XIV, 496 pages. 2005.

Vol. 3784: J. Tao, T. Tan, R.W. Picard (Eds.), Affective Computing and Intelligent Interaction. XIX, 1008 pages. 2005.

Vol. 3783: S. Qing, W. Mao, J. Lopez, G. Wang (Eds.), Information and Communications Security. XIV, 492 pages. 2005.

Vol. 3781: S.Z. Li, Z. Sun, T. Tan, S. Pankanti, G. Chollet, D. Zhang (Eds.), Advances in Biometric Person Authentication. XI, 250 pages. 2005.

Vol. 3780: K. Yi (Ed.), Programming Languages and Systems. XI, 435 pages. 2005.

Vol. 3779: H. Jin, D. Reed, W. Jiang (Eds.), Network and Parallel Computing. XV, 513 pages. 2005.

Vol. 3778: C. Atkinson, C. Bunse, H.-G. Gross, C. Peper (Eds.), Component-Based Software Development for Embedded Systems. VIII, 345 pages. 2005.

Vol. 3777: O.B. Lupanov, O.M. Kasim-Zade, A.V. Chaskin, K. Steinhöfel (Eds.), Stochastic Algorithms: Foundations and Applications. VIII, 239 pages. 2005.

Vol. 3776: S.K. Pal, S. Bandyopadhyay, S. Biswas (Eds.), Pattern Recognition and Machine Intelligence. XXIV, 808 pages. 2005.

Vol. 3775: J. Schönwälder, J. Serrat (Eds.), Ambient Networks. XIII, 281 pages. 2005.

Vol. 3774: G. Bierman, C. Koch (Eds.), Database Programming Languages. X, 295 pages. 2005.

Vol. 3773: A. Sanfeliu, M.L. Cortés (Eds.), Progress in Pattern Recognition, Image Analysis and Applications. XX, 1094 pages. 2005.

Vol. 3772: M. Consens, G. Navarro (Eds.), String Processing and Information Retrieval. XIV, 406 pages. 2005.

Vol. 3771: J.M.T. Romijn, G.P. Smith, J. van de Pol (Eds.), Integrated Formal Methods. XI, 407 pages. 2005.

Vol. 3770: J. Akoka, S.W. Liddle, I.-Y. Song, M. Bertolotto, I. Comyn-Wattiau, W.-J. van den Heuvel, M. Kolp, J. Trujillo, C. Kop, H.C. Mayr (Eds.), Perspectives in Conceptual Modeling. XXII, 476 pages. 2005.

Vol. 3769: D.A. Bader, M. Parashar, V. Sridhar, V.K. Prasanna (Eds.), High Performance Computing – HiPC 2005. XXVIII, 550 pages. 2005.

Vol. 3768: Y.-S. Ho, H.J. Kim (Eds.), Advances in Multimedia Information Processing - PCM 2005, Part II. XXVIII, 1088 pages. 2005.

Vol. 3767: Y.-S. Ho, H.J. Kim (Eds.), Advances in Multimedia Information Processing - PCM 2005, Part I. XXVIII, 1022 pages. 2005.

Vol. 3766: N. Sebe, M.S. Lew, T.S. Huang (Eds.), Computer Vision in Human-Computer Interaction. X, 231 pages. 2005.

Vol. 3765: Y. Liu, T. Jiang, C. Zhang (Eds.), Computer Vision for Biomedical Image Applications. X, 563 pages. 2005.

Vol. 3764: S. Tixeuil, T. Herman (Eds.), Self-Stabilizing Systems. VIII, 229 pages. 2005.

Vol. 3762: R. Meersman, Z. Tari, P. Herrero (Eds.), On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops. XXXI, 1228 pages. 2005.

Vol. 3761: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Part II. XXVII, 653 pages. 2005.

Vol. 3760: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Part I. XXVII, 921 pages. 2005.

Vol. 3759: G. Chen, Y. Pan, M. Guo, J. Lu (Eds.), Parallel and Distributed Processing and Applications - ISPA 2005 Workshops. XIII, 669 pages. 2005.

Vol. 3758: Y. Pan, D.-x. Chen, M. Guo, J. Cao, J.J. Dongarra (Eds.), Parallel and Distributed Processing and Applications. XXIII, 1162 pages. 2005.

Vol. 3757: A. Rangarajan, B. Vemuri, A.L. Yuille (Eds.), Energy Minimization Methods in Computer Vision and Pattern Recognition. XII, 666 pages. 2005.

Vol. 3756: J. Cao, W. Nejdl, M. Xu (Eds.), Advanced Parallel Processing Technologies. XIV, 526 pages. 2005.

Vol. 3754: J. Dalmau Royo, G. Hasegawa (Eds.), Management of Multimedia Networks and Services. XII, 384 pages. 2005.

Vol. 3753: O.F. Olsen, L.M.J. Florack, A. Kuijper (Eds.), Deep Structure, Singularities, and Computer Vision. X, 259 pages. 2005.

Vol. 3752: N. Paragios, O. Faugeras, T. Chan, C. Schnörr (Eds.), Variational, Geometric, and Level Set Methods in Computer Vision. XI, 369 pages. 2005.

Vol. 3751: T. Magedanz, E.R.M. Madeira, P. Dini (Eds.), Operations and Management in IP-Based Networks. X, 213 pages. 2005.

Vol. 3750: J.S. Duncan, G. Gerig (Eds.), Medical Image Computing and Computer-Assisted Intervention – MICCAI 2005, Part II. XL, 1018 pages. 2005.

Vol. 3749: J.S. Duncan, G. Gerig (Eds.), Medical Image Computing and Computer-Assisted Intervention – MICCAI 2005, Part I. XXXIX, 942 pages. 2005.

Preface

This volume contains the papers presented at the 11th workshop on Job Scheduling Strategies for Parallel Processing. The workshop was held in Boston, MA, on June 19, 2005, in conjunction with the 19th ACM International Conference on Supercomputing (ICS05).

The papers went through a complete review process, with the full version being read and evaluated by an average of five reviewers. We would like to thank the Program Committee members for their willingness to participate in this effort and their excellent, detailed reviews: Su-Hui Chiang, Walfredo Cirne, Allen Downey, Wolfgang Gentzsch, Allan Gottlieb, Moe Jette, Richard Lagerstrom, Virginia Lo, Jose Moreira, Bill Nitzberg, and Mark Squillante. We would also like to thank Sally Lee of MIT for her assistance in the organization of the workshop and the preparation of the pre-conference proceedings.

The papers in this volume cover a wide range of parallel architectures, from distributed grids, through clusters, to massively-parallel supercomputers. The diversity extends to application domains as well, from short, sequential tasks, through interdependent tasks and distributed animation rendering, to classical large-scale parallel workloads. In addition, the methods and metrics used for scheduling and evaluation include not only the usual performance and workload considerations, but also considerations such as security, fairness, and timezones. This wide range of topics attests to the continuing viability of job scheduling research.

The continued interest in this area is reflected by the longevity of this workshop, which has now reached its 11th consecutive year. The proceedings of previous workshops are available from Springer as LNCS volumes 949, 1162, 1291, 1459, 1659, 1911, 2221, 2537, 2862, and 3277 (and since 1998 they have also been available online).

Finally, we would like to give our warmest thanks to Dror Feitelson and Larry Rudolph, the founding co-organizers of the workshop. Their efforts to promote this field are evidenced by the continuing success of this workshop. Even though they are stepping down from the organization of the workshop, we hope they will continue to lend their expertise and contribution to the workshop and the field as a whole.

August 2005

Eitan Frachtenberg
Uwe Schwiegelshohn

Table of Contents

Modeling User Runtime Estimates

Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson

School of Computer Science and Engineering,
The Hebrew University, 91904 Jerusalem, Israel
{dants, etsman, feit}@cs.huji.ac.il

Abstract. User estimates of job runtimes have emerged as an important component of the workload on parallel machines, and can have a significant impact on how a scheduler treats different jobs, and thus on overall performance. It is therefore highly desirable to have a good model of the relationship between parallel jobs and their associated estimates. We construct such a model based on a detailed analysis of several workload traces. The model incorporates those features that are consistent in all of the logs, most notably the inherently modal nature of estimates (e.g. only 20 different values are used as estimates for about 90% of the jobs). We find that the behavior of users, as manifested through the estimate distributions, is remarkably similar across the different workload traces. Indeed, providing our model with only the maximal allowed estimate value, along with the percentage of jobs that have used it, yields results that are very similar to the original. The remaining difference (if any) is largely eliminated by providing information on one or two additional popular estimates. Consequently, in comparison to previous models, simulations that utilize our model are better in reproducing scheduling behavior similar to that observed when using real estimates.

1 Introduction

EASY Backfilling [19, 21] is probably the most commonly used method for scheduling parallel jobs at the present time [7]. The idea is simple: Whenever the system status changes (a new job arrives or a running job terminates), the scheduler scans the queue of waiting jobs in order of arrival. Upon reaching the first queued job that can not be started immediately (not enough free processors), the scheduler makes a reservation on the job's behalf. This is the earliest time in which enough free processors would accumulate and allow the job to run. The scheduler then continues to scan the queue looking for smaller jobs (require less processors) that have been waiting less, but can be started immediately without interfering with the reservation. The action of selecting smaller jobs for execution before their time is called *backfilling*.

To use backfilling, the scheduler must know in advance the length of each job, that is, how long jobs will run.[1] This information is used when computing the reservation time (requires knowing when processors of currently running

[1] This is true for any backfilling scheduler, not just EASY.

D. Feitelson et al. (Eds.): JSSPP 2005, LNCS 3834, pp. 1–35, 2005.
© Springer-Verlag Berlin Heidelberg 2005

jobs will become available) and when determining if a waiting job is eligible for backfilling (must be short enough so as not to interfere with the reservation). As this information is not generally available, users are required to provide runtime estimates for submitted jobs. Obviously, jobs that violate their estimates are killed. This is essential to insure that reservations are respected. Indeed, backfilling is largely based on the assumption that users would be motivated to provide accurate estimates, because jobs would have a better chance to backfill if the estimates are tight, but would be killed if the estimates are too short.

However, empirical investigations of this issue found that user runtime estimates are actually rather inaccurate [21]. Results from four different installations are shown in Fig. 1 (Section 4 discusses the four presented workloads in detail). These graphs are histograms of the estimation accuracy: what percentage of the requested time was actually used. The promising peak at 100% actually reflects jobs that reached their allocated time and were then killed by the system according to the backfilling rules. The hump near zero was conjectured to reflect jobs that failed on startup, based on the fact that all of them are very short (less than 90 seconds). The rest of the jobs, that actually ran successfully, have a rather flat uniform-like histogram.

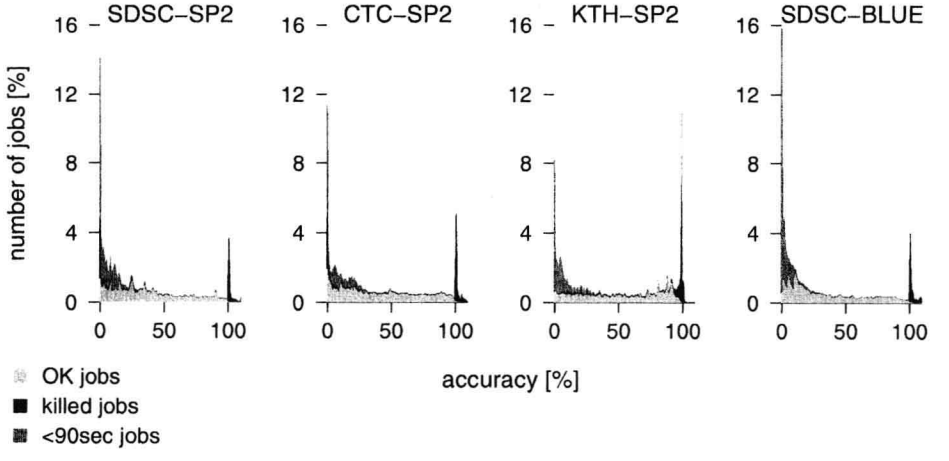

Fig. 1. Accuracy histogram of user runtime estimates: $accuracy = 100 \times \frac{runtime}{estimate}$

The issue of user runtime estimates has since become the focus of intensive research. A number of studies have suggested that inaccurate runtime estimates are actually good, as they provide the scheduler with more flexibility and eventually lead to better performance; as a result, it was even proposed to simply double the user runtime estimates before using them [29, 21], or further, randomizing them [22]. In contrast, other studies contend that accurate runtime estimates are actually better, as they can lead to even better performance if used correctly, e.g. by scheduling in some SJF (shortest job first) based order

此为试读,需要完整PDF请访问: www.ertongbook.com

[14, 23, 1, 25]. Still other studies have shown that the accuracy of user estimates can have non-trivial effects on the results of performance evaluations [8].

1.1 Motivation

All this activity spurred a search for ways to model user runtime estimates. Such a model is needed for three reasons. First, it is useful as part of a general workload model that can be used to study different job scheduling schemes, e.g. by means of simulation. Second, it is often the case that existing log files from production systems (used to drive simulations) are missing this information; a model can help in artificially manufacturing it. Third, a model may provide insights that will be useful in the study of whether and how the inaccuracy of estimates may be exploited by the scheduler.

We would like to make it clear that this paper targets the first two reasons mentioned above, that is, we aim to model and reflect reality, not to make it better. Indeed, in a different study, we show how backfilling schedulers can produce and utilize better runtime predictions that dramatically improve performance [25]. But even this novel technique often relies on user estimates under various conditions. Additionally, recall that user estimates have a role that is different than just serving as approximated runtimes, as they are also part of the user contract: the system guarantees a job will never be killed before its user estimate is reached. Consequently, system generated predictions (or other conceivable future mechanisms that are similar) can't "just" replace estimates.

At the same time, estimates ensure that jobs will indeed be killed at some point. Systems with no user estimates at all (that is, no runtime upper bound) are also undesirable, as these will allow jobs to run indefinitely, potentially overwhelming the system. At the very least we would expect users to choose some runtime upper-bound from a predefined set of values. However, this scenario is rather similar to reality, in which most users are already limiting themselves to very few canonical "round" estimates (as will be shown below), and jobs that exceed their estimates are immediately killed. It turns out there is actually no fundamental difference between allowing users to choose "any value", or from within a limited set.

Therefore, regardless of any possible scheduling improvements or changes, it seems a parallel workload model will not be complete if realistic user estimates are not included. Importantly, we will show that systems perform better if real user estimates are replaced with artificial ones, generated by existing models. This uncaptured "badness" quality of real user estimates constitutes a serious deficiency of existing models, as the purpose of these is to reflect reality, not to paint a brighter (false) picture. While counter intuitive, our goal in this paper is to produce estimates such that performance is worsened, not improved. Only when such a model is available, we can take the next step and consider ways to improve performance, based on a truly representative workload.

In the reminder of this section we survey the estimate models that have been proposed, and point out their shortcomings. This motivates the quest for a better model, which we propose in this paper.

1.2 Existing Models

The simplest possible model is to assume that user estimates are accurate. For example, such a model was used by Feitelson in [8]. This approach has two advantages: it is extremely simple, and it avoids the murky issue of how to model user estimates correctly. However, as witnessed by the data in Fig. 1, it is far from the truth.

A generalization of this model is to assume that a job's estimate is uniformly distributed within $[R, (f+1)R]$, where R is the job's runtime, and f is some non negative factor (f can't be negative because jobs are killed once their estimates are reached). If $f = 0$, this means that the estimates are identical to runtimes; if $f = 4$, they are distributed between R and $5R$, with an average of $3R$. Arguably, higher f values model increasingly inaccurate users. This model, which we call the "f-model", was proposed by Mu'alem and Feitelson [11] and several variants of it were used to investigate the effects of inaccuracy [29, 21, 1]. It was also used by several researchers in simulations using workloads that did not contain estimates data [13, 8]. The main problem with this model is that the estimates it creates are overly correlated with the real runtimes, so it actually gives the scheduler considerable amount of valuable information that is unavailable when real user estimates are used. In particular, it enables the scheduler to effectively identify shorter jobs and select them for backfilling, leading to SJF-like behavior. For example, under this model, a one-hour job will always appear longer than a one-minute job (in reality, this is often not the case). This leads to better performance results than those observed when using real user estimates.

A third model, also proposed by Mu'alem and Feitelson, attempts to reproduce the histograms of Fig. 1. These flat histograms imply that $R/E = u$, i.e. that the ratio of the actual runtime R to the estimate E can be modeled as a uniformly distributed random variable ($u \in [0, 1]$). By changing sides we find that given a runtime R divided by u results in an artificial estimate E. While unrelated to the actual user estimate for this particular job, this is expected to lead to the same general statistics of all the estimates taken together. The model also created the peak at 100% and the hump at low values. Finally, if E came out outrageous (because u happened to be very small), it was truncated to 24 hours. This was called the "ϕ-model" by Zhang et al. [27] (ϕ denoted the fraction of jobs in the 100% peak), who used it in various simulations.

The problem with this model is that it is missing a "hidden" factor which is often overlooked: that all production installations have a limit on the maximal allowed runtime. For example, on the SDSC SP2 machine this limit is 18 hours. Naturally, the limit also applies to estimates, as it is meaningless to estimate that a job will run for say 37 hours if all jobs are limited to 18 hours.

Consider Fig. 2 which displays the average accuracy of jobs grouped to 100 equally sized bins according to their runtime, for four different production traces. It has previously been conjectured that the apparent connection between longer runtimes and increased accuracy, is because the more a job progresses in its computation, the grater its chances become to reach successful completion [3]. However, this false hypothesis ignores the existence of a maximal allowed runtime,

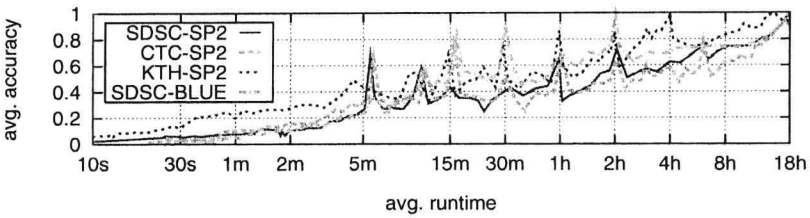

Fig. 2. Average accuracy as a function of jobs' (binned) average runtime

which suggests long jobs are guaranteed to have high accuracy. For example, if a job runs for 17 hours, its estimate must be in the range of 17 to 18 hours, so it's using at least 94.4% of its estimate. In other words, in contrast to the underlying assumption of the ϕ-model, the distribution of jobs in the accuracy histogram (Fig. 1) is not uniform. Rather, long jobs must be on the right, where accuracy is high, while short jobs tend to be on the left, at lower accuracies.

A fourth rather similar model was proposed by Cirne and Berman [3], which took the opposite direction in comparison to the previous model and chose to produce runtimes as multiples of estimates and accuracies, while generating direct models to the latter two. This decision was based on the argument that accuracies correlate with estimates less than they do with runtimes. In their model, accuracies were claimed to be well-modeled by a gamma distribution (this seems to be the result of trying to model the uniform part of the histogram along with the hump at low accuracies, by using one function for both). Estimates were successfully modeled by a log-uniform distribution. This methodology suffers from the same problem as the previous model, because accuracy is again independent of runtime. In addition, this model is not useful when attempting to add estimates to existing logs that lack them, or to workloads that are generated by other models which usually include runtimes and lack estimates [10, 6, 15, 20].

In addition to the per-model shortcomings mentioned above, there are two drawbacks from which all of them collectively suffer: The first is lack of repetitiveness: The work of users of parallel machines usually takes the form of bursts of very similar jobs, characterized as "sessions" [8, 28]. In the SDSC-SP2 log for example, the median value of the number of different estimates used by a user is only 3, which means most of the associated jobs look identical to the scheduler. It has been recently shown that such repetitiveness can have decisive effect on performance [26]. The second shortcoming is a direct result of the first: estimates form a modal distribution composed of very few values, a fact that is not reflected in any existing model. This is further discussed in the next section.

The conclusion from the above discussion is that all currently available models for generating user estimates are lacking in some respect. Consequently, using them in simulations leads to performance results that are generally unrealistically better than those obtained when real user estimates are used. Our goal in this paper is to capture the "badness" of real user estimates by finding a model that matches all known information about them: their distribution, their connection with each job's runtime, and their effect on scheduler performance.

2 Modality

We require a model capable of generating realistic user estimates. The usual manner in which such problems are tackled is by fitting observed data to well known distributions, later to be used for producing artificial data. To some extent, this methodology is applicable when modeling estimates, which appear to be well captured using the log-uniform distribution [3] as shown in Fig. 3.

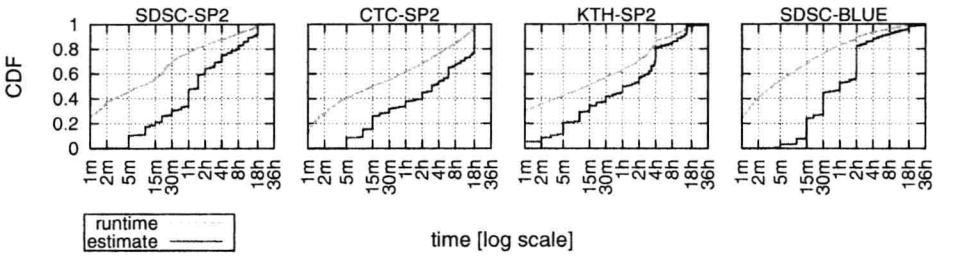

Fig. 3. Runtime and estimate CDFs (cumulative distribution functions) of the four workload traces. Runtime-curves are much higher than estimate-curves because runtimes are much shorter than estimates. For example, in CTC, 40% of the estimates are shorter than one hour (60% are longer), while for runtimes the situation is reversed (only 40% are longer than one hour).

The difficulty lies in that user estimates embody another important characteristic: unlike runtimes, they are inherently modal [21, 2, 17], because users tend to repeatedly use the same "round" values (e.g. five minutes, one hour, and so on). This is reflected in the staircase-like estimate curves of Fig. 3, in which each mode corresponds to a popular estimate value.

In particular, note the significant modes located at the maximal estimate of each trace, where the runtime and estimate curves finally meet (in Section 4 we will see that 4h and 2h effectively serve as the maximal estimate values of KTH-SP2 and SDSC-BLUE, respectively). Evidently, the maximal estimate is always a popular value. For example, this value is used by a remarkable 24% of CTC jobs. This phenomenon probably reflects users' lack of knowledge or inability to predict how long their jobs will run, along with their tendency to "play it safe" in the face of strict system policy to kill underestimated jobs.

In the context of job scheduling, this observation is quite significant, as maximal-estimate jobs are the "worst kind" of jobs in the eyes of a scheduler as they are too long to be backfilled. In fact, if all jobs chose their estimates to be the maximal value, all backfilling activity would stop completely.[2]

The observation about the maximal estimate mode may also be applied, to some extent, on other (shorter) modes: Consider the scenario in which an SJF

[2] Except for when using the "extra" nodes, see [21] for details.

scheduler must work with estimates that are highly inaccurate. If these estimates nevertheless result in a relatively correct ordering of waiting jobs, performance can be dramatically improved (up to an order of magnitude according to [1]). However, if estimates are modal, many jobs look the same in the eyes of the scheduler, which consequently fails to prioritize them correctly, and performance deteriorates. In general, if the estimate distribution is dominated by only a few large monolithic modes, performance is negatively effected, as less variance among jobs means less opportunities for the scheduler to perform backfilling.

Modality is absent from existing estimate models. An immediate heuristic that therefore comes to mind when trying to incorporate modality, is to "round" artificially generated estimates (e.g. by one of the models described above) to the nearest "canonical" value: values smaller than 1 hour are rounded to (say) the nearest multiple of 5 minutes, values smaller than 5 hours are rounded to the nearest hour, and so on. Experiments have shown that this heuristic fails in capturing the badness of user estimates, and performance results are similar to those obtained before this artificial modality was introduced. Additionally, arbitrary "rounding" fails to reproduce the various properties of the estimate distribution, as reported in the following sections.

The fact of the matter is that modes have a different (worse) nature than produced by the above. For example, when examining the number of jobs associated with the most popular estimates, we learn that these decay in an exponential manner e.g. half of the jobs use only 5 estimate values, 90% of the jobs use 20 estimates values etc. In contrast, the decay of less popular modes obeys a power law. In fact, almost every estimates-related aspect exhibit clear "model-able" (that can be modeled) characteristics.

3 Methodology

The modal nature of estimates motivates the following methodology. When examining a trace, we view its estimate distribution as a series of K modes given by $\{(t_i, p_i)\}_{i=1}^{K}$. Each pair (t_i, p_i) represents one mode, such that t_i is the estimate-value in seconds (t for time), and p_i is the percentage of jobs that use t_i as their estimate (p for percent or popularity). For example, the CTC mode series includes the pair $(18h, 23.8\%)$ because 23.8% of the jobs have used 18 hours as their estimate. Occasionally, we refer to modes as *bins* within the estimate histogram. Note that $\sum_{i=1}^{K} p_i = 100\%$ (we are considering all the jobs in the trace). The remainder of this section serves as a roadmap of this paper, describing step-by-step how the $\{(t_i, p_i)\}_{i=1}^{K}$ mode-series is constructed.

3.1 Roadmap of This Paper

Each of the following paragraphs correspond to a section or two in this paper, and may contain some associated definitions to be used later on.

Trace Files. We build our model carefully, one component at a time, in order to achieve the desired effect. Each step is based on analyzing user estimates in traces from various production machines, in an attempt to find invariants that are not unique to a single installation. The trace files we used and the manipulations we applied on them are discussed in Section 4.

Mass Disparity. Our first step is showing that the modes composing the mode-series naturally divide into two groups: About 20 "head" estimate values are used throughout the entire trace by about 90% of the jobs. The rest of the estimates are considered "tail" values. This subject is titled "mass disparity" and is discussed in Section 5. We will see that the two mode groups have distinctive characteristics and actually require a separate model. Naturally, the efforts we invest in modeling the two are proportional to the mass they entail.

Number of Estimates. We start the modeling in Section 6 by finding out how many different estimates there are, that is, modeling the value of K. Note that this mostly effects the tail as we already know the head size ($\sim$20).

Time Ranks. The next step is modeling the values themselves, that is, what exactly are the K time-values $\{t_i\}_{i=1}^{K}$. The indexing of this ascendingly sorted series is according to the values, with t_1 being the shortest and t_K being the maximal value allowed within the trace (also denoted T_{max}). The index i denotes the *time rank* of estimate t_i. This concept proved to be very helpful in our modeling efforts. We also define the *normalized time* of an estimate t_i to be t_i/T_{max} (a value between 0 and 1). Section 7 defines the function F_{tim} that gets i as input (time rank), and returns t_i (seconds).

Popularity Ranks. Likewise, we need to model the mode sizes / popularities / percentages: $\{p_j\}_{j=1}^{K}$. This series is sorted in order of decreasing popularity, so p_1 is the percentage of jobs associated with the most popular estimate. The index j denotes the *popularity rank* of the mode to which p_j belongs. For example, the popularity rank of 18h within CTC is 1 ($p_1 = 23.8\%$), as this is the most popular estimate. We also define the *normalized popularity rank* to be j/K (a value between 0 and 1). Section 8 defines the function F_{pop} that gets j as input (popularity rank), and returns p_j, the associated mode size.

Mapping. Given the above two series, we need to generate a mapping between them, namely, to determine the popularity p_j of any given estimate t_i, which are paired to form a mode. Section 9 defines the function F_{map} that gets i as input (time rank) and returns j as output (popularity rank). Using the two functions defined above, we can now associate each t_i with the appropriate p_j. This yields a complete description of the estimates distribution. The model is then briefly surveyed in Section 10.

Validation. Finally, the last part of this paper is validating that the resulting distribution resembles the reality. Additionally, we also verify through simulation that the "badness" of user estimates is successfully captured, by replacing the original estimates with those generated by our model. The replacement activity

此为试读,需要完整PDF请访问: www.ertongbook.com