Classification of Load Distribution Algorithms

Reinhard Riedl and Lutz Richter
Department of Computer Science
University of Zürich
Zürich, Switzerland, CH-8057

Abstract

The allocation of workload in distributed systems has almost as many views as one would like. The attempt to find suitable structures to classify the different ways of solving this task can therefore reflect the underlying model only. In the following paper a three models is developed. The load model is used to describe the workload and its current distribution within the system. The action model defines for a given algorithm at any instant of time the eligible next step(s). And finally, the solution model shows the formal context for obtaining and evaluating the load distribution decisions. All three models and their interrelationship are illustrated by selected examples of existing load distribution approaches. The result of the following investigation is the recommendation that load distribution algorithms can be classified according the five criteria: objectives, type and amount of used information, the source of the distribution, the parameter time, and the initiating instance.

1 Introduction

Aim and ambition of the following lines is to discuss different *adaptive load distribution algorithms for sequential execution of work in distributed systems* by way of a *specification and classification.* In the past there has been published an enormous amount of literature on such algorithms. Most of them were shown to work well in the environments and for the test cases presented by their authors. Little work, however, has been done to compare the qualities of different algorithms.

Therefore, we propose and discuss various criteria for the classification of adaptive load distribution algorithms, since we believe that detailed descriptions of existing algorithms according to these classification schemes will make such comparisons considerably easier. We believe that this is especially of importance to the customer who has to decide on the choice of the load distribution procedures. In particular, we suggest that future publications of new algorithms ought to be accompanied by their classification according to the main schemes discussed in the following.

Most load distribution algorithms presented in literature are only partial load distribution mechanisms, as they deal with particular aspects of load distribution and lack complete implementation in real life systems. Therefore, a main point of any classification ought to be a specification of the part of load distribution, which is covered by the algorithm, since this makes simulation results and analytic performance evaluations more transparent and their comparison easier.

In the following, we shall only deal with explicit load distribution activities of the distributed system. We shall ignore implicit load distribution, as e.g. data moving or system component performance control, system administrator activities and input control activities. Little work has been done on that so far. Moreover, we shall not deal with specific real time computing aspects either. For other classifications see e.g. [3] and [9].

2 Models and heuristics

There is nothing like a general model for distributed systems, which could yield analytic algorithms for load allocation. Even if there were such a general model available it would necessarily be too complicated for algorithmic treatment in run-time. Therefore, *heuristics* have to be used by any load allocation algorithm, in particular for the choice of the used formal models.

There is a hierachy of **formal models** involved in load distribution. Any load distribution mechanism relies on an architecture model which specifies the functions of the various system components. We shall not discuss this model here explicitely, but put emphasis on its reflection in the specific models affiliated with load distribution.

In that respect, first there is the model of the dynamic components of the distributed system considered by load distribution, which we shall call **load**

1066-6192/96 $5.00 © 1996 IEEE
Proceedings of PDP '96

model. It describes the *workload* the system has to deal with, comprises a formal description of runtime *system states* as they are considered by the load distribution mechanisms, and it describes the *information* about these states available at runtime. The description of the workload has to contain, among others, the definition of load units, relations among them, and their processing needs, and it has to contain the description of data units and their relations, accessed by the load units. The description of the system states and the description of available information may concern the state of system resources, the distribution of load units in the system, and the distribution of data with respect to load units – the latter usually in an implicit way due to the complex nature of that issue.

Seen from another point of view, the load model consists of the following three parts: a model of physical reality, a model of observations, and a model of relations among observations and reality (which describes how one can derive conclusions on load states from observations). For an algorithm which puts emphasis on the latter see e.g. [13]. In most cases the load model, which represents the *knowledge* of the load distribution and thus relies on heuristic choices of its designers, cannot completely be determined from the knowledge of the load distribution algorithm, if it is not explicitely supplied by its designers.

Then there is a model which defines the actions allowed in which situations to be taken by which component of the distributed system – due to hardware and software availabilities and possibly a heuristic choice of restrictions. We shall call that model **action model**. It deals with distribution actions *and* collecting information – and thus represents the *physical activities* of the load distribution – and it describes the *initiation* of any kind of decision making involved.

Thirdly, for that decision making, there is the model which sets the algorithmic formal context for the load distribution decisions as well as the decisions for information collecting. We shall call that model **solution model**. It primarily reflects the *objectives* which the load distribution algorithm is supposed to pursue, the *strategy* it is supposed to use for their pursuit, and the heuristic knowledge of the computing context and its relevant factors for system performance.

Note that so-called heuristic algorithms typically restrict their intuitive reasoning to the choice of a simple solution model (or even a trivial solution model in case the algorithm is completely determined by the action model). On the contrary, so-called analytic load distribution algorithm, impose heuristic assumptions at many different places. (For this reason, we shall not use an 'analytic versus heuristic' classification, which may often be found in the classification literature.)

In order to use the solution model for load distribution activities system state information has to be supplied. In some cases this supply is again subject to decisions based on runtime evaluation of a component of the solution model.

Only after the load model, the action model and the solution model have been fixed, there appear non-heuristic analytic solutions in the strict sense of the word. In order to distinguish between the solution model and the solutions obtained by the implementation of the algorithm in the context of the solution model we shall address the actual problem solving by the algorithm as the **solution making procedure**. Implementations may apply various heuristics for approximate solution of the formal problems set up by the solution model in order to save overhead, or the nature of the problem may exclude exact solutions. Thus the solution making procedure may determine the performance in a characteristic way. For example in [6] Newton's approximation method is used for the solution of the adaptive optimization problem of the solution model.

In principal there are three different tactics to make load distribution decisions, when objectives, strategies and heuristic system performance models are given by a load model – which might then indeed be used for classification: solving the formal problem analytically, either exactly or approximately, solving a simplified problem, or guessing the solution according to some heuristics, respectively. The solution of the formal problem in the context of the solution model may thus take very different forms, but these are only a secondary issue for our purposes compared with the load model, the action model and the solution model.

It follows from the above that there are overlaps between the three models quoted above, and there are logical and physical implications among them. Objectives both influence the choice of the action model and the solution model (and to some extent the choice of the load model, too). The action model relates to the load descriptions of the load model. The load model model both supplies static load characterizations and the formalism for dynamic system state descriptions to the solution model. And the solution model only considers actions defined by the load model.

Various heuristics identical in nature may appear at different stages of the load allocation procedure yielding different views and results. The analysis of al-

gorithms with respect to the application of heuristics may help to clarify algorithmic results and to avoid misunderstandings. We suggest that future presentation of new algorithms should be accompanied by an informal, but detailed description of which heuristics are applied in which model, in order to make the comparison of algorithmic qualities easier and research results more transparent.

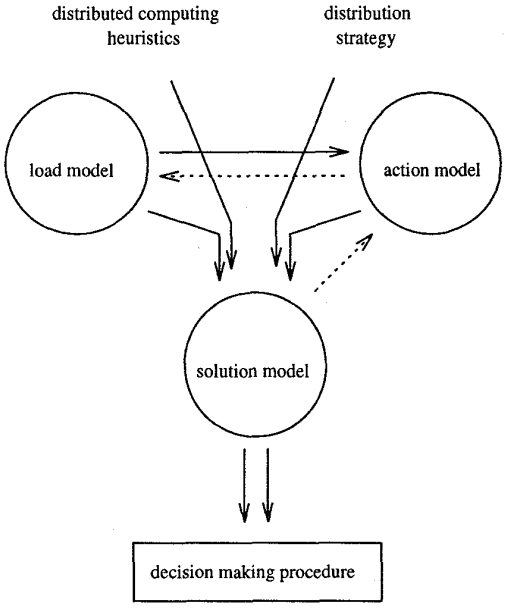

Figure 1

Figure 1 represents the various 'control flows' among the three models and the solution making procedure. The choice of the load model essentially restricts the possibilities for the choice of the solution model, as all objects subject to distribution appear in the solution model. (This does not relate to the process of the development of the load distribution algorithm, but only to the a-posteriori logical implications.) Therefore, there are also some implications in the opposite direction. The solution model makes use of the load model and the action model, and it reflects performance measurements of distributed computing in the system under consideration as well as the distribution strategy chosen by the designer. (Again, this only relates to the logical a-posteriori implications.) In the opposite direction, the solution model suggests the choice of the action model. Finally, the decision making procedure results from the implementation of all three models. It is the realization of the solution model.

2.1 Examples

As an example let us look at the algorithms discussed in [14]: They are based on the well known four-rules model, or equivalently four-policies model, with *information rule, transfer rule, location rule,* and *selection rule.* That model is concerned with fully distributed load-distribution, where nodes may either fetch load units from remote nodes or deliver them to remote nodes.

The load model underlying [14] assumes that all load units are undistinguishable, enter the system at any one of the nodes, and leave it after execution. Each load unit remembers the number of transfers it has experienced so far. Moreover, the mean service times for load units at each particular node are known, too. Further stochastic assumption on the input and the load execution, which are made to describe the simulation model, do not concern the algorithm itself.

A system state at runtime (of the N-nodes system) is represented as an (N+1)-dimensional vector, whose first N entries are the numbers of load units residing at each node, and whose last entry is the number of load units in the communication system. There is only local knowledge of the local entry to this vector available for load distribution plus – in one version of the algorithm – additional knowledge of the loads units having been transfered from a remote node in the recently past time interval of some fixed length. For that version the system state is described by the above N-dimensional vector plus a record of all transfers of load units in the time window.

The action model contains only fully distributed (i.e. local) activities: local-remote (one-way) transfer of load units, local-remote (both ways) communication for the collection of information, local decision making and a local triggering of decision making. Transfer is restricted to that load unit which arrived most recently, but more than one transfer per unit is allowed. The arrival of a load unit triggers, in that order, decision making according to the transfer rule, the information rule and the location rule (where the latter two are only used in case that an attempt to transfer a load unit was decided).

In a general four-rules model, the information rule describes, when and how to collect and where to store information used in decision making. Its activities are described by that part of the action model concerned with the collection of information, while the information rule itself is that component of the solution model concerned with information collection. Moreover, the triggering of decision making with respect to the col-

lection of information, as it is described in the action model, is defined by the information rule, too.

In [14] three distinct information rules are evaluated, each of which assumes that the collection of information takes place at each node separately triggered by the decision to attempt to transfer a load unit and that it is performed only at instances when a node has decided locally to transfer one of its tasks. Additionally, one of the three algorithms, LBT, keeps track of the tasks which it has received from other nodes. The part of the solution model concerned with the collection of information either demands information from all remote nodes, or, in the other two versions, from a fixed number of remote nodes. In the latter case it considers all nodes as equally eligible for a load transfer, except for the LBT–version, which restricts attention to nodes, which did not send a task to the local node in the recent past, and it chooses the nodes by random choice – i.e. the final decision making is performed as random polling from a set of nodes specified by the solution model.

In general the transfer rule describes when to initiate an attempt to transfer a load unit, and whether or not to perform it, finally. It is affiliated to that component of the action model concerned with the triggering of decision making and to that component of the solution model, which deals with distribution decisions. Used in that four-rules approach, it implies that the set of load distribution activities defined in the action model contains only local-remote load unit transfers.

In [14] the transfer rule defines the initiation of decision making by the arrival of a new task. Decisions (on whether or not to attempt a load transfer) are made by the comparison of the local load state with a fixed local threshold value (depending on the assumed mean local servive time and a general choice of the system administrator).

In general, the location rule defines how to choose the partner for the load transfer. It is thus affiliated with the solution model.

In [14], first the node with the lowest virtual load (among those nodes which the local node communicated with about their load state) is calculated. Then the difference δ of its virtual load with the local virtual node is calculated. Finally that δ is compared with the number of times the 'triggering load unit' has been transfered so far, and a decision is taken. Decision making affiliated with location policy is thus completely deterministic. In general, the selection rule is used to select a task for the transfer. It is thus affiliated with the solution model.

In [14] the selection rule is already fixed by the restriction of the action model to transfer only the 'triggering units'.

As another example, let us consider the algorithm presented in [10], which involves a stochastic learning automaton. The load units in its load model are again all undistinguishable. The system state (of the N-nodes system) is described by a an (N+1)-dimensional vector, whose first N entries take the values *underloaded / overloaded*, while the last entry is itself an N-dimensional vector, whose entries define the locally (at each node) observed global system load state, taking values in $\{0, 1, ..., N-1\}$. System information for load distribution is only locally available as a 2-dimensional vector whose first entry is the local load state and whose second entry is the locally observed global system state. The latter information is considered to differ from reality, which is the basis for the usage of a stochastic learning automaton – otherwise the whole concept would fail. At regular instances node to node communication exchanges the first component of the local load vector, and the recalculation of the second entry of the local load vector takes place at each node independently, after the node has obtained information from all other nodes.

Thus the action model contains node to node communication about load states at regular instances, and the local triggering of the computation of the system wide load state. Moreover, it contains the local triggering of the local decision making when a new load unit arrives, and local-remote transfer of load units. The resulting load distribution is fully distributed and sender-initiated.

The solution model is made up of two parts: the one described above concerned with the local calculation of the global system state, and the one which is concerned with the decision where to execute a load unit. The latter is a stochastic learning automaton, which calculates load distribution matrices using feedback about the success or failure of a transfer to a remote node (dependent on whether the chosen remote node was underloaded or overloaded). It consists itself of two parts: distribution according to a probability law and the following update of the probability law upon feedback.

The decision making involves three procedures: straightforward calculations of locally observed global load states, throwing the coin according to a probability distribution in order to decide where to execute a load unit, and a straightforward calculation of the update of the probability vector using remote feedback

information.

So far, we have not yet specified the meaning of overloaded, respectively underloaded load states, and we shall revisit the algorithm for that pupose in the following. There is one ambiguity in the specification of the algorithm with respect to the load model, the action model and the solution model. Information is passed from one decision making procedure to the next as distribution matrix, whose rows correspond to system wide load states. Thus, more precisely, the information about the system state available for load distribution consists of the 2-dimensional vector described above plus the local distribution matrix, and consequently, the global load state is described by the above (N+1)-dimensional vector plus an N-dimensional, whose entries are distribution matrices.

The choice of the two examples above does not imply any statement whatsoever about the quality, importance or originality of the cited papers. Both load distribution mechanisms have been discussed in the literature various times. We chose them as they enabled us to demonstrate various different features of our models.

Finally, we would like to emphasize that although there are no purely analytic algorithms, load distribution cannot successfully be based on purely heuristic reasoning only. Quite to the contrary, since the choice of models and heuristics more or less determines success and failure of load distribution, that choice ought to be based on analytic reasoning.

In the following chapters we shall discuss various choices of load, action and solution models and the resulting classifications, some of which have been indicated in the examples above. For comparisons of various different distribution algorithms see e.g. [15].

3 Objectives

Objectives are an important issue for classification of algorithms. There are two ways, how objectives may appear in the solution model (and we shall only consider objectives, which appear explicitely in the solution model): as *cost functions* and as *input variables to the solution model.*

As cost functions are necessarily heuristics only, objectives related to them cannot always realized with certainty. Obviously, in general it is impossible to attribute a unique objective to a cost function, as various different objectives considered by load distribution algorithms are positively correlated. Moreover, the attributes may depend on load state and other variables as well. Analysis in the language of fuzziness might therefore help in the future to demonstrate implicit correlations among different objectives and different cost functions. Results of the analysis of correlations in turn may be used for the design of fuzzy decision machines, which are extensions of decision machines as presented in [13], and make use of load dependent performance of algorithms with respect to objectives on the one hand, and fulfil sensible users' needs on the other hand.

Another natural way how objectives determine the performance of load distribution algorithms are input variables to the solution model (cf. [10]). This is characteristic of models without explicitely formulated cost functions, where control of quantities to be optimized takes place.

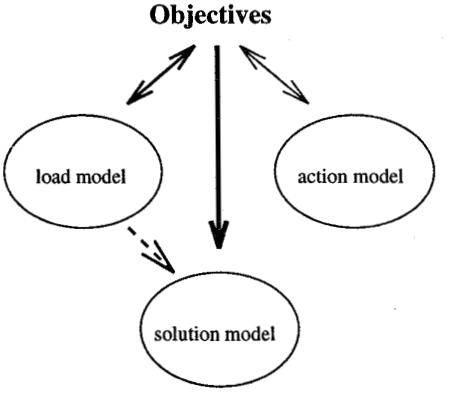

Figure 2

Primary objectives are

- 'good' service performance, as observed by tasks:
  - high performance in terms of
    - short waiting times
    - short service times
    - short elapsed (response) times
  - predictability of elapsed (response) times
  - availability of services
  - reliability of the system
- small (physical) distances between tasks and data, i.e. 'high' affinity
- 'good' service performance, as observed by resources of the system:
  - high global performance, in particular high throughput

  - high local performance, for example in the view of few cache misses or short communication times

- balanced situations, in particular
  - equallized distribution of load
  - adjusted usage of available resources
- 'good' utilization of resources:
  - small idle times of resources, or to the contrary
  - little usage of resources, e.g. little traffic

Fairness or *preferential treatment of special task classes* are secondary objectives of load allocation (which may be defined as a boundary condition or as a weighting factor in a cost function). Some of the above objectives are negatively correlated for special load situations. Others are only weakly positively correlated as they deal with different functions of the same quantity: e.g. availability and throughput. Evaluation of algorithms may reveal orientation towards some quantitative load situations or some qualitative load situations with respect to data access patterns of the load, which in turn might be used for classification too.

Figure 2 indicates how objectives are related with the choice of the three models. As we have explained above they may appear in the solution model explicitely via cost functions and implicitely via observables defined in the load model. Moreover, the choice of the action model is somewhat influenced by the objectives. On the contrary, the load model and the action model set boundary conditions for the pursuit of the objectives.

On the one hand any objective may be formulated in terms of a (non-unique) cost function. On the other hand, most algorithms which do not make use of a cost function explicitely, may be reformulated in terms of cost functions – for the price of leaving the (local) context of the solution model as it was set by the load model (and allowing solutions to be taken only from a small set).

As an example for the latter case let us consider the algorithms in [10]: There the costs for a distribution decision at node k are the sum of the entries of an N-dimensional cost vector. Its i-th entry is the product of the i-th entry of the global system state vector and the i-th entry in the row of the decision making matrix with the number found as the k-th entry of the N+1st coordinate of the global system state vector. Consequently, the decision making is the update of one row of the system matrix in order to reduce the costs. Hereby only updating operations are allowed, which add a reward vector whose entries are functions of the corresponding entry in the row to be updated and the cooresponding entry in the global system state vector.

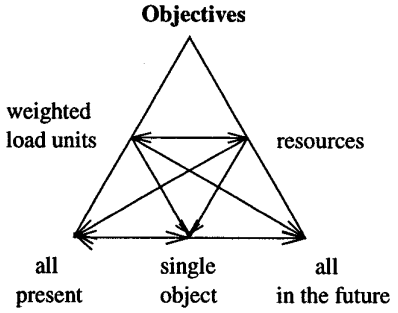

Figure 3

The pursuit of objectives may depend dynamically on load states. We distinguish between algorithms with

- various different strategies,
- a single strategy algorithms but dynamic parameter tuning
- a static single strategy

For the second case see e.g. [1]. Case 1 may be realized with the usage of (fuzzy) decision machines. (It appears to some extent in [10], where load distribution depends on observed load states.) The cost function in the solution model may either consider *local or global performance* of tasks or resources as it is depicted in Figure 3. In the latter case, this may be done with or without inclusion of *future workload*, and tasks may be attributed weights. The comparison of different algorithms ought to contain their detailed specifation in that respect. For example, these weights may be positively or negatively correlated to the complexity of the tasks. (FCFS-disciplines correspond to a weighting according to which each task is worth its execution time.)

4 Information usage

4.1 Suppliers and users of information

Part of the load model is the definition of

- which source supplies information in runtime and
- what kind of information is supplied (in runtime), while part of the solution model is the definition of

- which component exploits the supplied information, i.e. which component carries out the decision making procedure.

There are two distinct ways how information about the system's load state may be obtained:

1. it is taken from measurement communicated to the load distribution source
2. it is obtained from conjectures differing from the communicated information, either because available information is incomplete, or it is assumed to differ from physical reality.

In [13] Bayesian analysis is applied in order to calculate the expectation of performance as a function of the observed system state (assuming that observation and physical reality are correlated but differ in general). A so-called decision machine is used for that purpose, which makes use of a formal load model listing the possible states together with their probabilities and the possible observations together with the conditional probabilities of states with respect to observations, a formal action model listing the possible actions, and a formal solution model containing an utility function as cost function. However, it is important to observe that the corresponding load model, action model and solution model of the algorithm discussed in [13] *differ* from the models of the decision machine (but contain them as *submodels*). In particular monitor nodes are involved for regular updates of the conditional probabilities, which moreover vary from node to node. In [12] regression-based algorithms are discussed, which rely on the idea that looking at the past enables one to predict the future.

This leads us to the distinction between solution models which use past decisions of the same decision making instance as input and those who do not. [10] is another example of the first ones, [14] is an example of the latter ones. As we have shown above, in the first case the runtime system state description is increased by the corresponding description of that information. *LBT* in [14] is a special case, though not a rare case: there information about load distribution decisions at other places in the system is used as input for the decision making procedure – in form of some version of vector time stamps; cf. [11].

There is information used about current states and past performance of resources, about distribution of tasks in the system and past performance of tasks the system, and about distribution of data throughout the system. The influence of the type of information used on the performance of the algorithm has been investigated in [10]. There the load model describes the system state with the terms overloaded and underloaded. The author compared algorithms which used different interpretations of these terms, all of which were resource oriented: waiting queue lengths, size of free available memory, CPU usage, combinations of these, etc. As the author indicates, the algorithm itself was designed with task oriented intentions, namely reduction of response times. However, as response times does not appear explicitely in the solution model, it is not one of the objectives according to our definition..

Locality of data accesses plays a major part in load distribution. That phenomenon is usually addressed by the term *affinity*. More generally, affinity addresses the fact that certain tasks have preferences for special executing nodes, or may even be executed exclusively at special nodes, due to the distribution of data and tools. See e.g. [7].

Algorithms may use information both explicitely and implicitely. A special case of implicit usage is *feedback* of a posteriori judgement of decisions for the purpose of adaption of decision strategies, namely *'primitive learning'* algorithms, as for example the algorithm in [10].

Classification of algorithms with respect to their information usage may be based on distinction between usage of global and usage of local information, on distinction between algorithms with automatic supply of information as input data for the solution model, algorithms with separated solution models for information supply and solution models which jointly consider information supply and load distribution, and on distinction of algorithms with explicit and with implicit information usage. However, there are also algorithms which comprise, in the one or other respect, both alternatives. The various possibilities are depicted in Figure 4.

4.2 Static versus dynamic approaches

The easiest load models, as the corresponding solution models, do not consider any information about the current workload state of the system. We shall call such algorithms

- static distribution algorithms, contrary to
- dynamic algorithms.

Naturally, there are various intermediate forms between completely static and completely dynamic algorithms: e.g. static algorithms which are initialized

by some dynamic device reacting to changes in the workload state, or combinations of static and dynamic components.

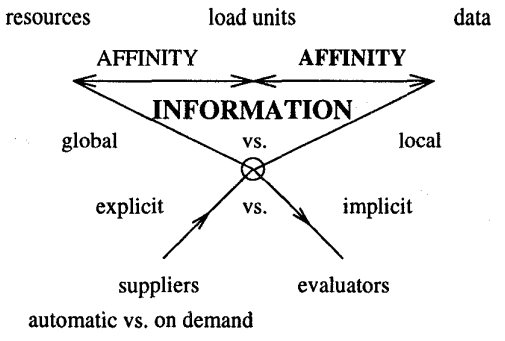

Figure 4

4.3 Affinity scheduling

Differences of data access times essentially determine the performance of distributed computing. This may be considered for the distribution of both tasks and resources.

In order to deal with the problem of data accesses, affinity scheduling is applied. This is already reflected on the level of the load model by an affiliation of a vector to each load unit which describes the execution costs at each of the nodes, mainly reflecting the need for data accesses. On the level of the solution model affinity appears in terms of cost functions, which result from the affinities stated in the load model.

There are various physical factors which set up physical affinity:

- the distribution of data and resource needs for the various task classes
- the degree of static activity initialized by a task
- data and task migration times
- the actual current cache consistencies created by the load distribution algorithms

Only in rare cases, there is no affinity assumed by the load and the solution model. By its very nature physical affinity can only be observed partially, implicitely and with some observation delay. A rough but useful classification of algorithms is the distinction into those, whose load models describe task-data relations, and those which do not. Obviously, affinity scheduling relying on task-data affinities presupposes an appropriate workload analysis. If preprocessing is too expensive, it requires the characterization of load units by a-priori observables which preferably take only values from a small set, and which are strongly positively related with the data accesses of the load unit. For a discussion of workload characterization and further literature see [2] or [5].

5 The action model: initiators, source of distribution and time

A primary classification of load distribution may be based on the *objects of distribution*, and a corresponding secondary classification on the *granularity* of these objects. Algorithms may distribute *tasks, resources* and *data* in various granulations. Two essential properties of the action model are the definition of

- which source initiates load distribution and when it is performed, and
- where it is performed: i.e.
  - which component is in charge of it
  - which sources supply (partial) services;

Another way to classify load distribution algorithms is to distinguish between algorithms with or without (time or event driven) redistribution. (This is of particular importance with respect to thrashing phenomena.)

5.1 Central versus distributed load distribution

As with respect to the source of distribution there are two different approaches: *central load distribution* versus *distributed load distribution.* There is also an intermediary concept: *semicentral load distribution* by a small subset of nodes, each of which is responsible for some rayon.

When discussing and comparing these concepts, distinction has to be made between *logical* and *physical* concepts. In most but not all cases, logical concepts agree with physical concepts. Naturally, physically central algorithms are also logically central, the inverse, however, must not necessarily hold. LINDA, [4], is an example of a logically central service which is implemented in a physically distributed way.

Moreover, one has to distinguish between the mechanism in charge of load distribution, and the services used for load distribution.

Naturally, there are various types of combinations of central, semicentral and distributed mechanisms. Distributed distribution may further be specified according to the range of load exchange partners for each node.

5.2 Sender Initiated versus Receiver Initiated Load Distribution

A classical criterion for the classification of load distributiom algorithms on the level of the action model is their partition into *sender-initiated* and *receiver-initiated* load distribution algorithms. However, it is hard to define a sharp distinction between both concepts, since in most cases sender-initiated mechanisms cooperate with receiver-initiated mechanisms, or sender data are taken into consideration by receiver-initiated mechanisms, and vice versa. In order to be more precise, we distinguish between algorithms which are (essentially)

1. central sender-initiated
2. distributed sender-initiated or receiver-initiated and fully distributed
3. receiver-initiated with a central source of load units

Here, the two cases in 2 exhibit rather similar behaviour. From an analytic point of view, the essential difference between central sender-initiated and distributed receiver-initiated load distribution algorithms which deal with a central input source is the time when distribution takes place. Resulting performance deficits for sender-initiated algorithms are an open mathematical question. The following two subdivisions are related with that time difference: sender-initiated upon-arrival distribution versus sender-initiated distribution with (load state dependent) time delay, and work-conserving versus not work conserving receiver-initiated load distribution. Here work-conserving means that no node will decide to stay idle in case that there are load units in the system without server. Algorithms, which are not work-conserving, are a special form of affinity scheduling. Little work has been done so far on that subject, but somewhat related results exist in the literature on anomalies in task scheduling.

Both sender-initiated and receiver-initiated algorithms do not necessarily use system state information. An example of a central sender-initiated algorithm is *static random load distribution* according to a probability law which only reflects the computing power of each node, see e.g. [8], where steady state optimization is discussed. Receiver-initiated algorithms are necessarily logically distributed, but they may use central services, which may in turn be implemented either at a central place or in a distributed manner, as was indicated above.

On the level of the load model one may classify algorithms with respect to the information about senders and receivers they use. (It is possible to implement a central sender-initiated algorithm which only uses information about receivers: e.g. a central job dispatcher which regularly checks soiurce states.)

6 Probabilistic concepts

Probabilistic concepts play an essential role in load distribution. They may enter as characteristic properties of the load model or of the solution model. Various different quantities may be considered as stochastic processes or variables: e.g. the workload, the current or future states of resources, the knowledge of the current physical state or the future input, or the behaviour of load distributors distinct from the local ones. In the decision making procedure stochastic reasoning may be used to filter rare cases out and to speed up some algorithms. Uncertainty of knowledge about the system's state may be treated by Bayesian analysis. Randomization is used to avoid unpleasant superposition and interference of load patterns.

Algorithms which are based on stochastic workload models do not necessarily apply stochastic solution models and vice versa. The specification and classification of algorithms may thus be based on the place where probabilistic reasoning enters (cf. Figure 5). In particular we distinguish between

- workload models with or without stochastic assumptions on the future
- stochastic and deterministic system state models
- algorithms using deterministic decisions and those applying probability distributions for random decision

These types of classification may substantially improve the understanding of an algorithm. In [6], the solution model suggests random distribution according to a distribution variable, whose value results from a deterministic solution problem, while the implemented solution making procedure approximates that value by a choice from the set $\{0, 1\}$, leading to deterministic load distribution. A feedback of this simplifi-

cation in the decision making procedure into the solution model would lead to a simplified solution model and in turn a simplified solution making procedure (which may be also be derived from formula (5) there).

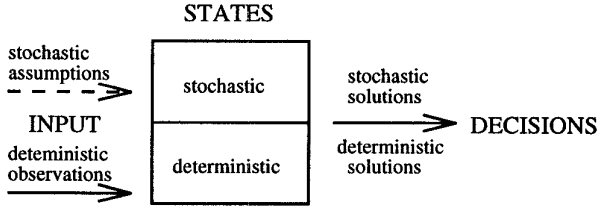

Figure 5

7 Concluding remarks

A primary classification of load allocation algorithms ought to be concerned with the following five issues: *objectives, information usage, source(s) of distribution, time,* and *initiators.*

It results from the above discussion that load distribution algorithms may be specified according to the introduced three models: the load model, the action model and the solution model together with the implemented decision making procedure. There are two major classification lines with a wide range of applicability: *usage of heuristics* and *application of affinity scheduling,* whereby affinity is understood in the most general sense and all kinds of affinities between tasks, data resources and processing resources are considered.

We believe that future work ought to be done on the relations between the three models presented above. In particular a detailed specification of cost functions and an analysis of their nature might prove to be helpful for the understanding of the load distribution issue in distributed computing.

References

[1] Avritzer A. et al.: The Advantage of Dynamic Tuning in Distributed Asymmetric Systems, Proc. Infocom '90, pp. 811 – 818, (1990)

[2] Born E., Delica T., Richter L., Riedl R.: Characterization of Workload for Distributed Processing – Methodology and Guide, Deliverable WP.2/T2.1 ESPRIT III P8144 (1995);

[3] Casavant T.L., Kuhl J.G.: A Taxonomy of Scheduling in General-Purpose Distributed Computing Sustems, IEEE Trans. Softw. Eng. 14, pp. 141 – 154, (1988)

[4] Carriero N., Gelernter D., and Ahuija S., Linda and Friends, IEEE Computer, 19(8), pp. 26 – 34, (1986)

[5] Calzarossa M., Serazzi G.: Workload characterization: a survey, Proc. IEEE, Vol. 81(8), pp. 1136 – 1150, (1993)

[6] Gelenbe E., Kushwaha R., Dynamic Load Balancing in Distributed Systems, Mascots '94, IEEE Computer Society Press, pp. 245 – 249, (1994)

[7] Haldar S., Subramanian D.K.: An affinity-based dynamic load balancing protocol for distributed transaction processing systems, Performance Evaluation 17(1), pp. 53 – 71, (1993)

[8] Kim C., Kameda H.: An Algorithm for Optimal Static Load Balancing in Distributed Computer Systems, IEEE Trans. Comp., Vol. 41(3), pp. 381 – 384, (1992)

[9] Kremien O., Kramer J.: Methodical Analysis of Adaptive Load Sharing Algorithms, IEEE Trans. Par. Distr. Syst. 3, pp. 747 – 760, (1992)

[10] Kunz T.: The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme' IEEE Trans. Softw. Eng. 17, pp. 725 – 730, (1991)

[11] Lamport L.: Time, Clocks, and the Ordering of Events in a Distributed System, Communications ACM, Vol 21(7), pp. 558 – 556, (1978)

[12] Leff A., Yu P.S.: A Performance Study of Robust Distributed Load Sharing Strategies, IEEE Trans. Parall. Distrib. Systems, Vol. 5(12), pp. 1286 – 1301, (1994)

[13] Stankovic J.A.: An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling, Trans. Comp. C-34, pp. 117 – 130, (1985)

[14] Zhang Y., Kameda H., Shimizu K.,: Adaptive bidding load balancing algorithms in heterogeneous distributed systems, Mascots '94, IEEE Computer Society Press, pp. 250 – 262, (1994)

[15] Zhou S.: A Trace-Driven Simulation Study of Dynamic Load Balancing, IEEE Trans. Softw. Eng. 14, pp. 1327 – 1341, (1988)