修士論文

省電力型データベース問い合わせ実行方式の研究

平成19年2月2日 提出

指導教員： 喜連川 優 教授

情報理工学系研究科 電子情報学専攻

56404 上野 裕也

目 次

図 目 次

概要

従来、コンピュータシステムにおける消費電力に関しては、主にモバイルコンピューティングにおけるプロセッサの省電力化を中心に検討がなされてきた。近年では、データセンタなどの大規模システムにおいてもその消費電力が問題となりつつあり、また、プロセッサだけでなく周辺の入出力機器を含めたシステム全体の省電力化が求められるようになっている。特に、システムの管理するデータ量が急激に増大している中、ディスクドライブの省電力化は極めて重要な課題である。

本論文では、多数のディスクドライブから構成されるディスクアレイの省電力化を目指し、データベースシステムの有する問い合わせ実行計画を利用した新しいディスクドライブの制御方式を提案する。独自の方式により構築したディスクドライブに関する消費電力モデルを示すとともに、当該モデルに基づく解析的検討により、従来方式と比較して大きな効果が得られることを明らかにする。

第1章　はじめに

1.1　動機

ディスクアレイは、サーバやデータセンタ等で利用されるコンピュータシステムを構成する主要デバイスであり、近年のデータ量の増大に伴い、その消費電力は極めて大きくなってきている。ディスクアレイの省電力に関する研究は、そのような流れの中でごく最近に始まったものである [6][7]。過去の研究の多くはラップトップマシンのディスクに関するものがほとんどで、ディスクアレイとは事情を異にするものであった。これらの背景として、以下のような要因が挙げられる。

まず、ネットワークの広帯域化によって、画像や、動画、音楽など、データ密度の大きな情報が広くやり取りされるようになったことで、データベースに蓄積されるデータの絶対量が増加していることが挙げられる。その増加傾向は現在も止まることを知らず、半導体におけるムーアの法則を上回るペースで増大している。更に、それらのデータを長期にわたり保存することを要請する法律の出現などによって、その管理費は非常に大きなものになっている。

また、現在のストレージの役割は、データの保存よりもむしろ管理に重きを置かれるようになった。例えば、広帯域を利用したデータのストリーミングを提供するサービスや、災害時の即時的なリカバリが求められるようになってきた [17]。万一災害時にリカバリが遅れてしまった時、場合によっては企業の倒産に繋がる恐れもあり、それに至らなかったとしても、時間に比例した莫大な損失が発生することは、現在の情報、通信の価値を考えれば想像に難くはない。このようなデータの管理コストの増大に伴って、ディスクアレイの省電力化が研究されるようになってきた。

最先端研究の一例として、Hibernator[1] という手法が挙げられる。これは、アクセスパターンのローカリティを利用してディスクの high speed と low speed (low power) を切り替えるものであり、データの migration を利用してディスクへのアクセス頻度をコントロールすることで比較的大きな粒度での効率化を実現している。ここでの効率とは、より高いパフォーマンスを保ちながら、より大きな削減電力を得ることを指す。

また、近年登場したストレージ製品として、MAID[2][3][4][5] が挙げられる。これは、最大で全てのディスクの約 4 分の 1 のみをアクティブにし、残りのディスクはアクセスが来た時のみスピンアップするという方法で消費電力を抑えている。ディスクストレージとテープストレージの両者の長所を抽出した特長を持っており、ローカリティの高いワークロードに適している。

またそれらを踏まえ、過去の研究では網羅できなかった部分を考察し、更に広い角度からの省電力へのアプローチを考えた上で研究の方向性を示す。

1.2 研究の目的

これまでの方式では、ストレージが省電力に関する制御を行う際、サーバから独立に動作していた。具体的には、ストレージはidle時間の長さやアクセス頻度など、ストレージの枠を出ない範囲の統計的な情報のみから、回転数変更などの省電力に関わる動作を行っていた。しかし、本研究では、アプリケーション、DBMSの持つ問い合わせ実行計画の情報を有効利用し、上位のレイヤから能動的にディスクの動作を決定するという従来とは異なる方式によって、ディスクアレイシステム全体としての消費電力削減を目指す。

更に、問い合わせの演算の中でも特に重い処理であるとされるJoinの最中に、アクセスされないと分かっているディスクを停止させることにより省電力を得る。Joinの中でも、 Hash Joinはこのような意図にとって有利な特徴を持っている。本研究では、このHash Joinを用いてディスクの省電力化を目指す。

1.3 論文の流れ

本論文の流れは以下の通りである。

まず、第2章で関連研究を紹介する。第3章でRDBとJoinに関する基本的な説明を行う。第4章で提案する方式の概要を示す。第5章で実験環境と、使用するディスクのスループットに対する電力モデルを示す。第6章で提案するモデル、並びに省電力化方式の評価を示し、最後に第7章にて論文のまとめと今後の課題を示す。

第2章　関連研究

2.1　spin speedのコントロール

ラップトップマシンのディスクに関する研究でもそうであったが、ディスクの省電力化の基本はディスクの不必要な時に回転数を低くするというものである。また、これらの実験においては、実際のサーバやデータセンタのI/Oリクエストをトレースしたものを入力として用いるが、それらのトレースにはローカリティ、つまり、データアクセスの偏りがある。ワークロードの種類にもよるが、ある調査では、新規にwriteされたデータのうち、50 %は再びreadされることがなく、30 %はたった一度のreadアクセスしかないという報告もある。このように、実際のワークロードにおいてはローカリティが顕著に現れ、これを利用して一部のディスクのみをfull speedで稼働させている傍らで、idle状態のディスクの回転を低下、または停止させて電力を削減するということが可能になる。

しかし、ここで問題になるのは、full speedのディスクをstandby状態にする時、あるいは逆にstandby状態のディスクをfull speedにする時は、余分な電力と遅延時間がかかってしまうということである。従って、あまり頻繁にディスクをon/offするという行為は逆に、電力とパフォーマンス両方面への性能低下を招いてしまう。更にそれだけにはとどまらず、ディスクの寿命にも影響が現れる。一般に、ディスクは数万回のspin on/offに耐えることができると言われているが、現実的な寿命を得るためにはspin on/offを一日数十回程度に抑える必要があり、あまり過度にspin speedを変えるというわけには行かない。

次に、図2.1を用いてディスクのspin speedをシフトするメカニズムについて説明する。まず、これまでのラップトップディスクに関する研究では、full speedとstandby状態を遷移して消費電力を抑えるというものであった。しかし、最近の研究では、full speed状態に加えて、更に低いspin speedにおいても読み書きできるようなmulti-speedディスク[10]を想定して、遷移にかかる時間的、電力的コストを抑える方法も見られるようになってきた。

更に、standby状態を介さないで直接spin speedを変えられるようなディスクを想定した研究もあり、良い成果を挙げているようであったが、このようなディスクも同様に未だ存在しない。これらのディスクを想定した評価では、実際にspin speedの違う二種類のディスクを用意するか、あるいはシミュレータを用いるなどの方法で実験を行っていた。従って、このようなディスクを実現することができれば、ディスクの省電力分野での大きな突破口となると思われる。

ラップトップディスクではspin speedを変えるタイミングのみがパラメータとなるが、ディスクアレイの場合はそれに加えてどのディスクのspin speedを変更するか、更に先述のようなmulti-speedのディスクを想定する場合にはどの程度のspin speedにするかというパラメータが加わる。これらのパラメータを決定するアルゴリズムを3つ紹介する。[1]

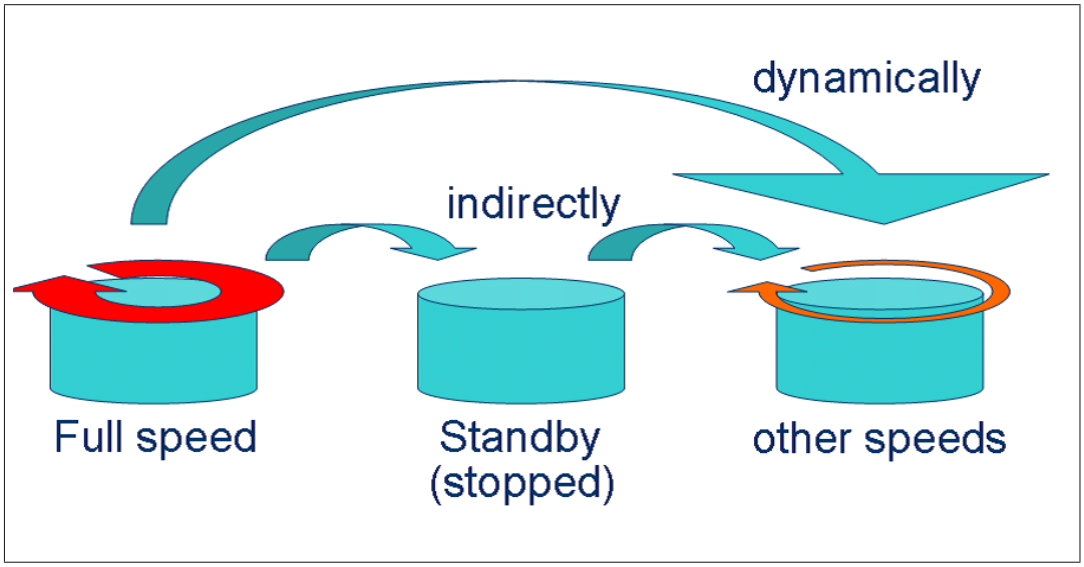

図 2.1: spin speed の遷移

Load Directed(LD) はスループットの何%かを閾値として、それより大きい負荷では spin speed を上げ、閾値より小さい負荷では spin speed を下げるという手法であり、段階を経て spin speed を変化させるためダイナミックな変化には対応できず、また、ディスクの寿命の観点からもこの特徴はあまり好ましいものとはいえない。

Dynamic RPM(DRPM) は各ディスクのキューに並んでいるリクエストの数と、リクエストの平均応答時間によって spin speed を変える方法である。平均応答時間の変化が閾値より大きければ spin speed を上げ、小さければ下げる。また、平均応答時間が上昇してきて閾値を超えたら、全てのディスクを full speed にすることでパフォーマンスを保証している。しかし、これも段階的な変化であり、また、キューのリクエスト数という指標ではワークロードの変化に鋭敏に対応できず、平均応答時間が上昇してしまうことが多かった。

Coarse-grain Response(CR) は数時間ごとという粗い粒度で、各ディスクに対するアクセス頻度を基準にして spin speed を決定する手法である。アクセス頻度の予測を行って直接的に spin speed を決定し、また、spin speed シフトの粒度が大きいので、ディスクの寿命にも影響を与えない。更に、予測が失敗し、パフォーマンスが低下した時にディスクを full speed にしてパフォーマンスを保証する手法も独立して取り入れることができる。

2.2 Hibernator

上記の特徴に基づき、近年発表された興味深い論文に、Hibernator[1] がある。

Hibernate とは冬眠を意味していて、リクエストのない時はディスクを文字通り休止させて消費電力を抑えるという手法である。Hibernator の特徴として、まず図 2.2 に示した tier という概念がある。これは要するに、「同じスピードで動作するディスクのグループ」を意味していて、

いくつかの tier がそれぞれのスピードで動作している。そして、tier を構成するそれぞれのディスクは、都合に合わせて tier 間を移動する、つまり、動作スピードを変えることができる。これは先ほど紹介した CR というアルゴリズムによってなされるもので、数時間に一回という粗い粒度で見直される。

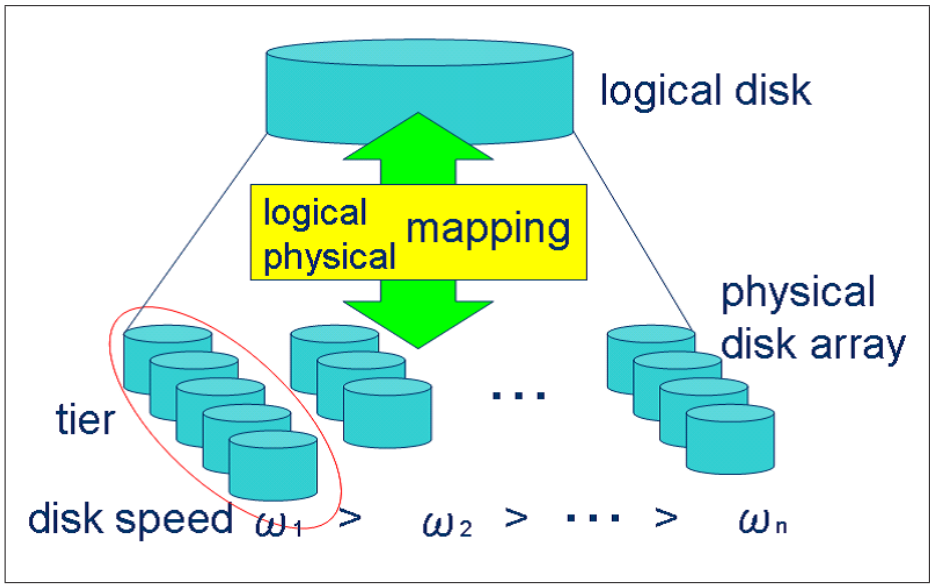

図 2.2: Hibernator における tier の構成

では、これらの tier がどのような特徴を持っていることが望ましいかを考えてみると、まず第一に、「spin speed の高い tier にはアクセス頻度の高いデータブロックが入っていること」が望ましいと考えられる。なぜなら、より多くのリクエストが高いスループットのサービスを得られるからである。

そして第二に、同じ tier の中の各ディスクへのアクセス頻度は均等になっている方がよいと考えられる。これは、一つのディスクにアクセス頻度の高いデータが集中すると、そのディスクのリクエストキューが長くなってしまって、仮に全てのディスクが full speed で動作していたとしても、このディスクがボトルネックとなって大きな遅延が発生してしまうことになるからである。従って、アクセス頻度は tier 内の各ディスク単位で見たときに均等になるように配置されていることが望ましいと考えられる。ちなみに Hibernator では、アクセス頻度は temperature と呼ばれる指標によってブロック単位で管理されている。実際には、このブロックの分散は Randomized Shuffling と呼ばれる、ランダムでパリティも含めてブロックを分散させる方法を取っている。下手にアクセス頻度を計算して配置するよりも効率よく分散できることが示されている。

Hibernator では、このような二つの特徴を保つようにデータの migration が行われている。また、workload の挙動が変化してこれらのバランスが崩れ、応答時間が著しく低下する場合がある。例えば、突然 low speed の tier のディスクに対するアクセスが集中し始めた、といった場合である。そのときは、全てのディスクを full speed にすることでパフォーマンスを保証している。

しかし、アクセスの絶対数が多くなってきた時、full speed の tier の中にあるディスクの数が

足りなくなってくることがある。そんな時は、CR アルゴリズムによって、数時間に一度 tier の構成が見直される。このとき、新たに full speed の tier に追加されたディスクのデータのアクセス頻度は周りと比べて小さいので、Randomized Shuffling を用いてディスク単位のアクセス頻度を均等にする。このような操作を foreground のリクエストがない時に、background 処理として行うことによって、応答時間に影響することなく migration が可能になる。

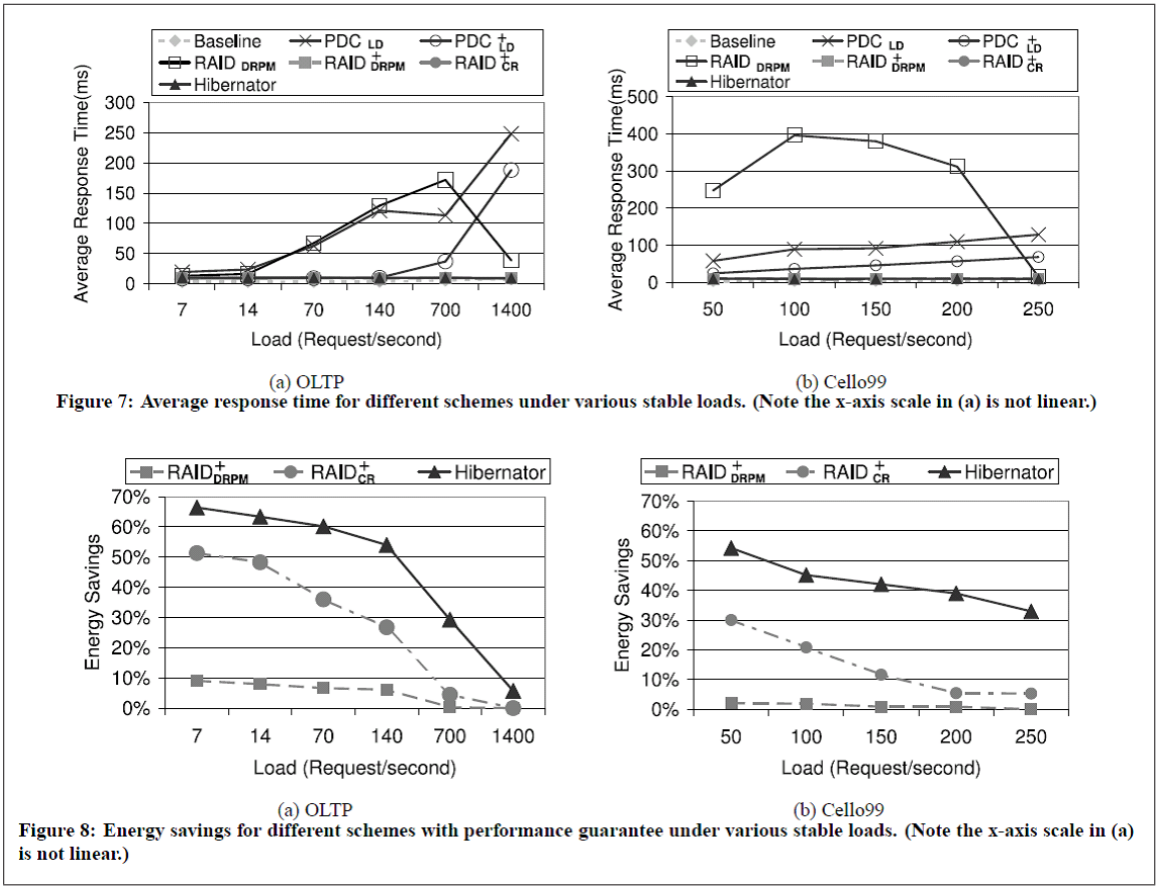

図 2.3: OLTP、Cello における Hibernator の平均応答時間と削減電力

図 2.3 は、OLTP、Cello99 におけるシミュレーションの結果を示したものであり、平均応答時間と削減電力がグラフになっている。アクセス頻度の高いディスクを一つのディスクに集中させる PDC の手法では、そのディスクがボトルネックになってしまい、今回のような比較的重いワークロードでは大きな遅延が生じてしまっているのがわかる。また、遅延が生じた時にディスクを full speed で回転させてパフォーマンスを保証する performance guarantee (PG) を用いていない RAID と DRPM の組み合わせも、同様に遅延が生じている。＋の印がついた PG を用いているものは到着頻度に関わらず応答時間が低く保たれており、PG を含め、全体としてそれぞれうまく働いていることがわかる。

更に、削減電力のグラフを見てみると、PG を用いた DRPM はあまりよい削減電力を見せていないが、PG を用いた CR と、更に migration も行っている Hibernator ではかなり良い削減電

力となっている。これらの違いとしては、ディスクの speed を見直して、spin speed をシフトする間隔が、DRPM では短いのに対し、CR ではかなり長い時間となっていることである。あまり粗い粒度での見直しはパフォーマンス低下に繋がる恐れがあるが、このグラフを見る限りそのようなことにはなっていないので、数時間に一度という見直しと migration が良い結果として現れたといえる。

2.3 その他の技術

spin speed をシフトする以外にも、省電力、効率化に繋がる技術が存在する。まず、データの migration が挙げられる。migration とは、データをディスク間で移動させることであり、migration によって I/O に都合がいいようにデータを再配置することによって効率化を実現できる。具体的には、ブロック単位で管理されたアクセス頻度によって、高速、低速のディスクにデータを割り振り、論理アドレスをマッピングし、効率のよいディスクアクセスを得る。

アクセスのローカリティが高い場合は、キャッシング [9] も有効になる。MAID[2][3][4][5] という製品ではこれを利用して消費電力を削減している。また、データの信頼性を上げる手法としては、RAID や HP の AutoRAID[8] といったものがある。AutoRAID とは、ミラーとパリティを組み合わせて、更に、data migration などの工夫を加えたものである。

以上のように、これまでの研究では、トランザクションのパターンにより、ディスクの動作アルゴリズムや閾値を考えるという方向性が主流となっていた。しかし、本研究では、クエリの問い合わせ実行計画を DBMS が工夫することにより、アクセスする必要のあるディスク、その順番等を判断し、能動的にディスクを制御するという状況を想定し、効率的な省電力化方式を提案している。

第3章　RDBとHash Join

今日、様々な分野で用いられているデータベースのほとんどは、関係データベース (以下 RDB と呼ぶ) と呼ばれており、独自の構造を持っている。本章では RDB の基本構造と具体的な動作を示し、本実験で重要な役割を果たす Hash Join に関して詳しく述べる。

3.1　RDBの構造

RDB は一つ以上のテーブルによって構成されている。テーブルは、複数の列、行によってマトリックス状になっており、列にあたる部分をアトリビュート、行にあたる部分をタプルと呼ぶ。

それぞれのアトリビュートには、データベースとして記録しておく事象の属性が割り当てられており、その属性に関する具体的なデータが記録されている。また、それぞれのタプルは、実世界における個々の実体に対応している。

例えば、図 3.1 では、テーブル “students” において、矢印の指すアトリビュート “class” にはそれぞれの生徒が所属するクラスが記録されており、矢印の指すタプルには Bob という生徒に関する情報が記録されている。

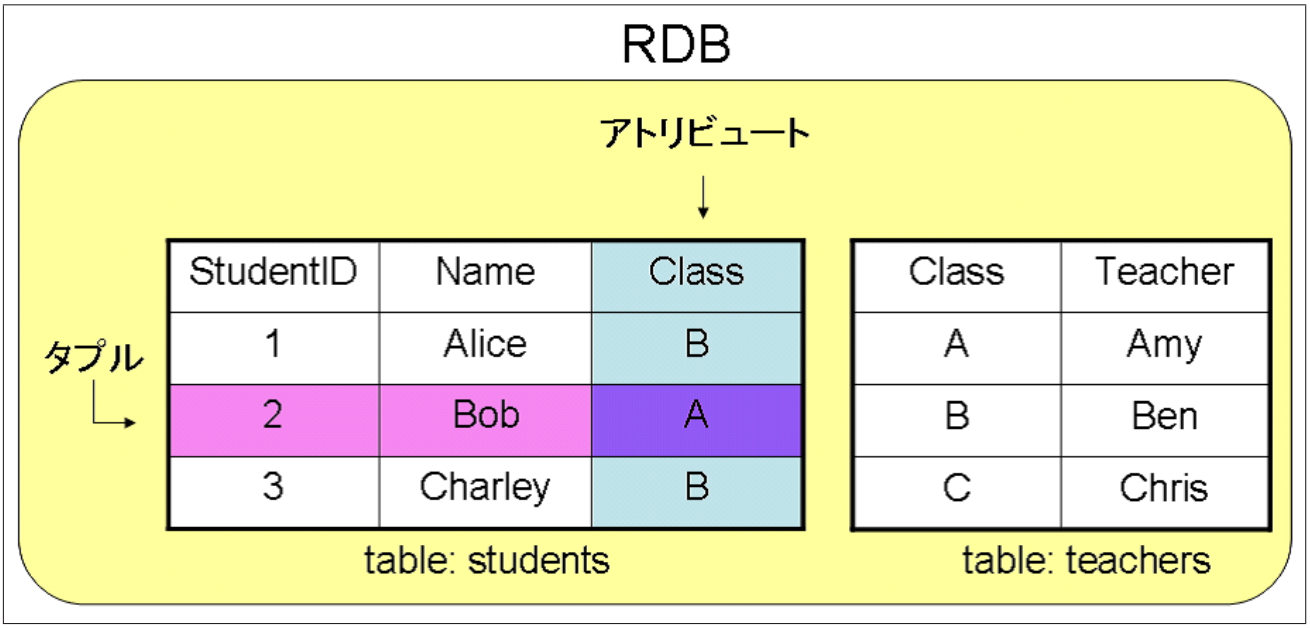

図 3.1: RDB の基本構造

3.2 問い合わせと演算

オープンソースである PostgreSQL[12]、MySQL[11] や、Oracle 社の Oracle、HITACHI 社の HiRDB、IBM 社の DB2 に代表される、RDB management system(以下 DBMS と呼ぶ) と呼ばれるソフトでは、図 3.1 のような形式のデータを管理している。

クライアントがこのようなデータベースから何らかの情報を得たいとき、Structured Query Language[1] (以下 SQL と呼ぶ) と呼ばれる問い合わせ言語によって、DBMS に対して問い合わせを行う。

例えば、「StudentID が 3 番の生徒の名前を知りたい」、「Class A に所属する生徒全員を知りたい」、「職員全員の名前を知りたい」、「生徒を担当する職員の名前を全て知りたい」など、様々な形の問い合わせが考えられる。DBMS は、SQL 独自の構文で特定の条件を与えられることにより、これらの問い合わせに対する答えを算出する機能を備えている。

「StudentID が 3 番の生徒の名前を知りたい」というときは、図 3.2 のような SQL 文を与える。すると、DBMS は、テーブル “students” の中から StudentID が 3 であるタプルを抽出し、アトリビュート “StudentID” と “Name” を出力していることが分かる。このような演算を選択演算と呼ぶ。

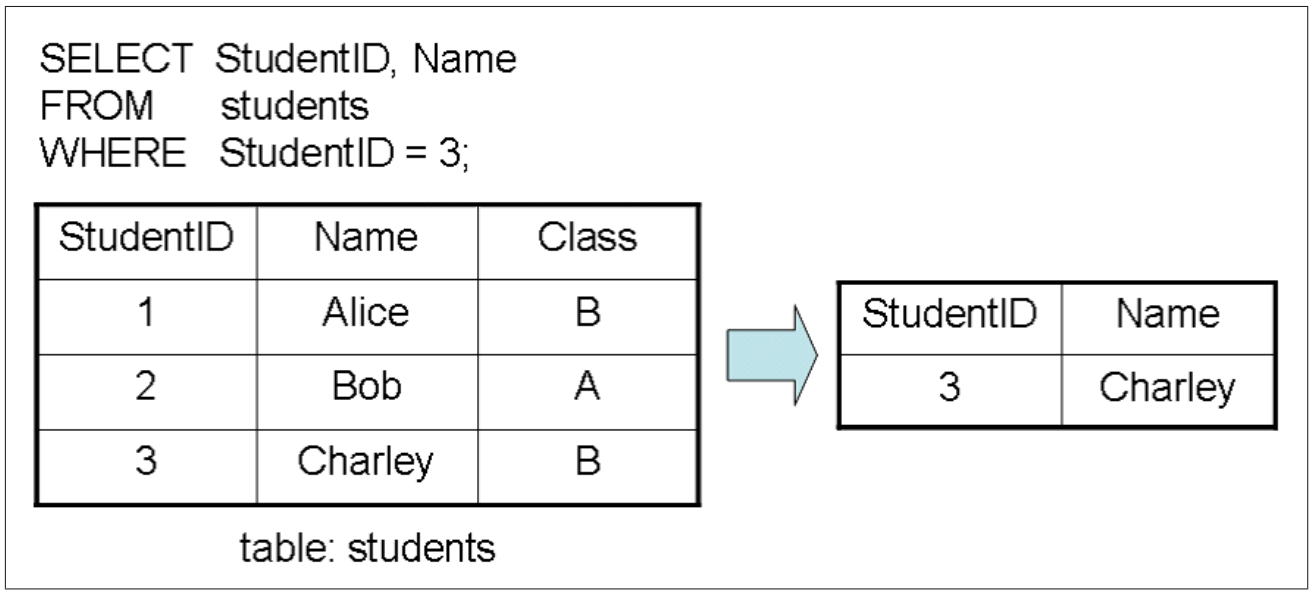

図 3.2: 選択演算

更に、「職員全員の名前を知りたい」というときには、図 3.3 のような SQL 文を与える。すると、DBMS は、テーブル “teachers” の中からアトリビュート “Teacher” を全て抽出し、出力する。このような演算を射影演算と呼ぶ。

[1] この表現は特に、IBM 社の DBMS の SQL を指すが、標準 SQL を指す表現のデファクトスタンダードとなっている。

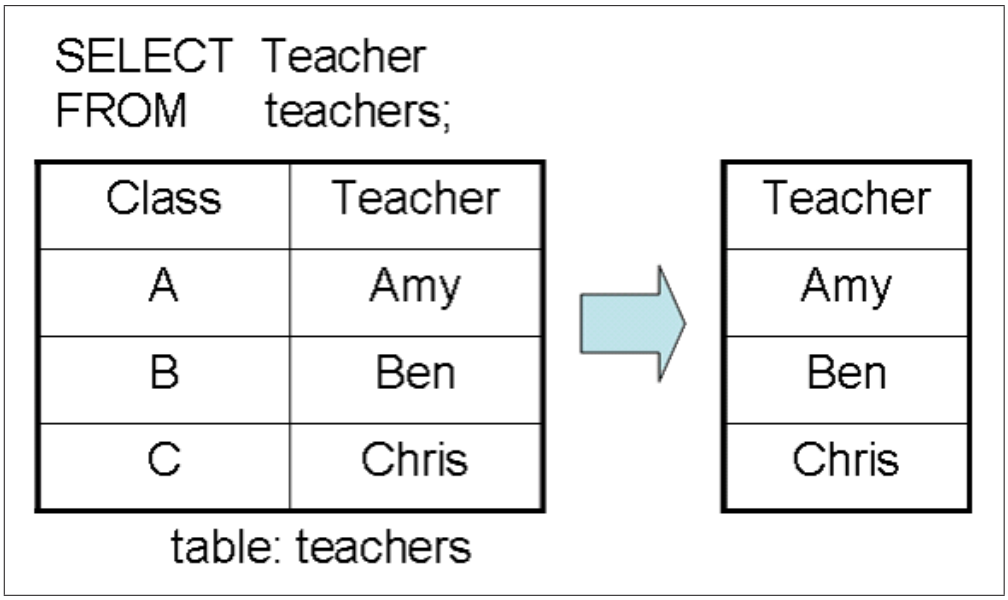

図 3.3: 射影演算

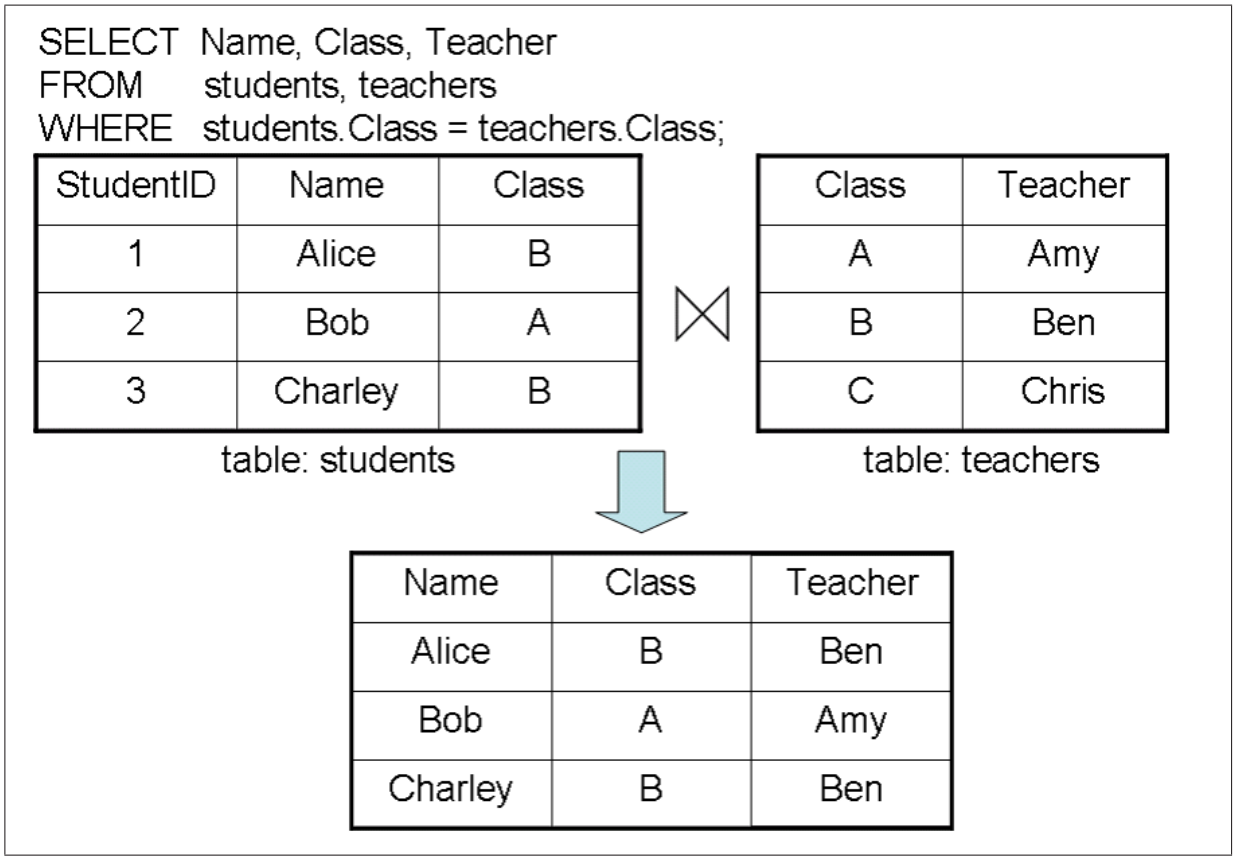

図 3.4: 結合演算

最後に、「生徒を担当する職員の名前を全て知りたい」というときには、テーブル students と、テーブル teachers のどちらか片方だけでは解答を得ることができない。このとき、両方のテーブルを同時に参照し、両者を組み合わせることで解答を得る。図 3.4 のような SQL 文を与えると、DBMS は、テーブル “students” とテーブル “teachers” の双方に共通のアトリビュート “Class” を走査して一致しているタプルを抽出し、アトリビュート “Name” と “Teacher” を出力する。このような演算を結合演算 (Join) といい、本研究において最も重要な演算となる。

3.3 Join の種類

Join とは、二つのテーブルのあるアトリビュートを比較する条件を課すことによってテーブルを結合する演算であり、問い合わせ処理の中でもっとも計算コストのかかる演算であるといわれている。なぜなら、条件に合うタプルを抽出するために、両テーブルの全タプルを走査しなくてはならないからである。

Join には、Nested Loop Join、Sort and Marge Join、Hash Join の三種類が存在する。

Nested Loop Join は、両テーブルの走査ループを入れ子 (Nested) にすることにより、タプルの全組み合わせで比較を行うというものであり、最も単純だが計算量も最も多い。

Sort and Marge Join は、比較対象となるアトリビュートの値がソートされている (またはソートする) ことが条件となる。それを前提にすると、以下のような流れで比較を行うことができる。まずテーブル A の始めのタプルを基準とし、テーブル B を順に走査して一致するものを抽出する。そして、走査していく中で、テーブル B での値が基準としたテーブル A のタプルの対象値を上回るものが現れた時[2]、今度はテーブル B のそのタプルを基準とし、テーブル A を走査する。これを繰り返すことにより、計算量を削減することができる。

最後に、Hash Join に関して説明する。Hash Join は、まず片方のテーブルを全走査し、ハッシュテーブルを作成する。次に、もう一つのテーブルの各タプルを同じハッシュ関数にかけ、作成したテーブルにつき合わせて一致するものを抽出するものであり、十分なメモリ領域を有するシステムにおいては非常に有効な手段となる。

[2]昇順でソートされている場合。降順ならば下回った時となる。

3.4 right deep と left deep

図 3.5 は、二つのテーブルを Hash Join する場合を木の構造で図示したものである。これを実行プラン木と呼ぶ。実行プラン木において、Join は ⋈ という記号で表す。

Hash Join を行う際、作業を二つのフェーズに分けることができる。図 3.5 を見ると、左のテーブルでハッシュテーブルを作成し、右のテーブルでハッシュテーブルとの一致を確認している。前者を build、後者を probe と呼ぶ。

build と probe、それぞれの動作が一つのタプルに対して行う操作は、build ではハッシュ関数にかけたあと、メモリ上のハッシュテーブルに追加するだけだが、probe ではハッシュ関数にかけたあと、ハッシュテーブルの中から一致するものを探し、更にハッシュ値が同じだが違うタプルである可能性を確認する操作が必要である。従って、一般的に build よりも probe の方が重い動作であるが、メモリのスワップなどが起こると build にもディスク I/O の分だけ遅延が生じることもある。

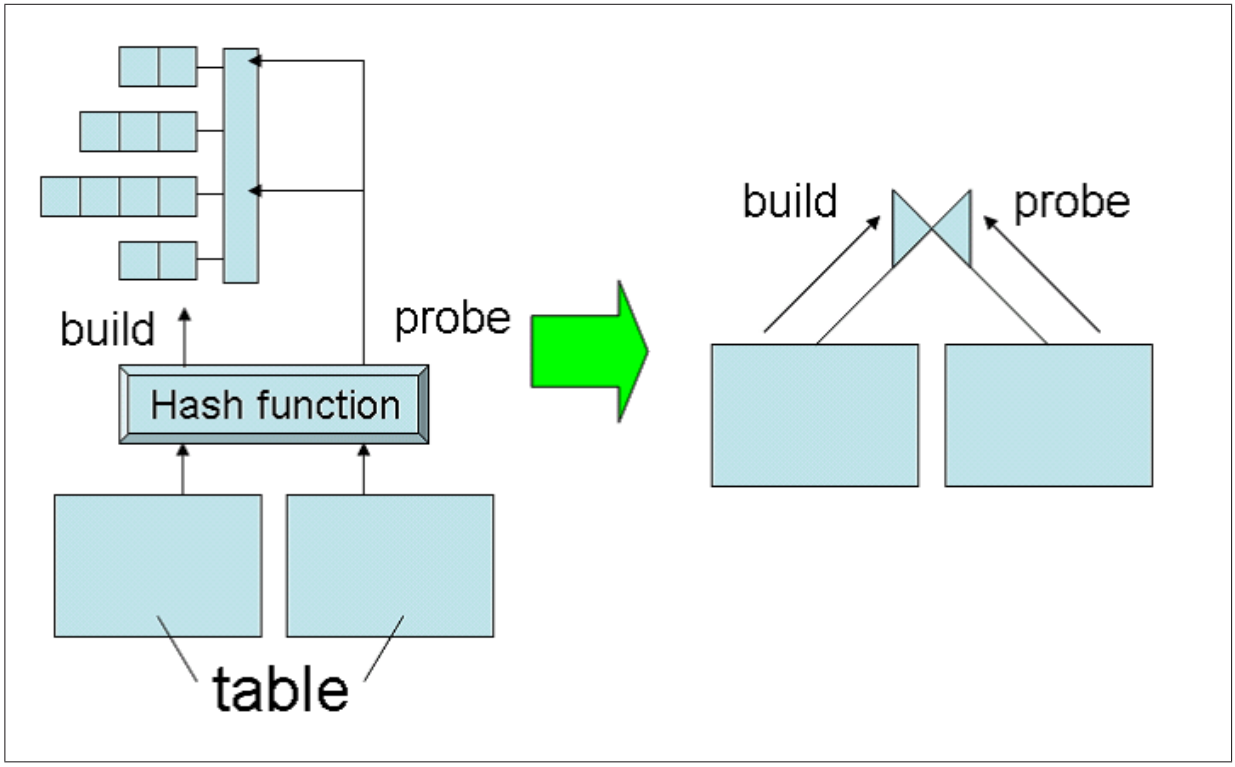

図 3.5: Hash Join

図 3.5 の右側のような実行プラン木の構造を図示する場合には、build するテーブルを左、probe するテーブルを右に描くようにする。従って、複数のテーブルを Hash Join する際には、どのテーブルで build を行い、どのテーブルで probe を行うか、また、どのような順番で build と probe を行うかによって、木の構造が変わってくる。

また、木の構造が変わると、それぞれの段階の Hash Join でどれほどの Hit があるかが変わ

り、後のハッシュテーブルの大きさも変わってくるので、従って全体のスループットも変わってくる。

例えば、図 3.6 を見てみると、木の左側に位置しているテーブル全てで build を行いあらかじめハッシュテーブルを作成しておき、一番右下のテーブルで一度に probe を行っている。このような構造の実行プラン木を right deep と呼ぶ。

このとき、build を行うテーブルは順不同であり、DBMS によって最適化されるが、ディスクの内側にあるものから読み込んでしまうことが多い。また、メモリ領域が十分でないと、スワップが起こり十分なスループットは発揮できない。

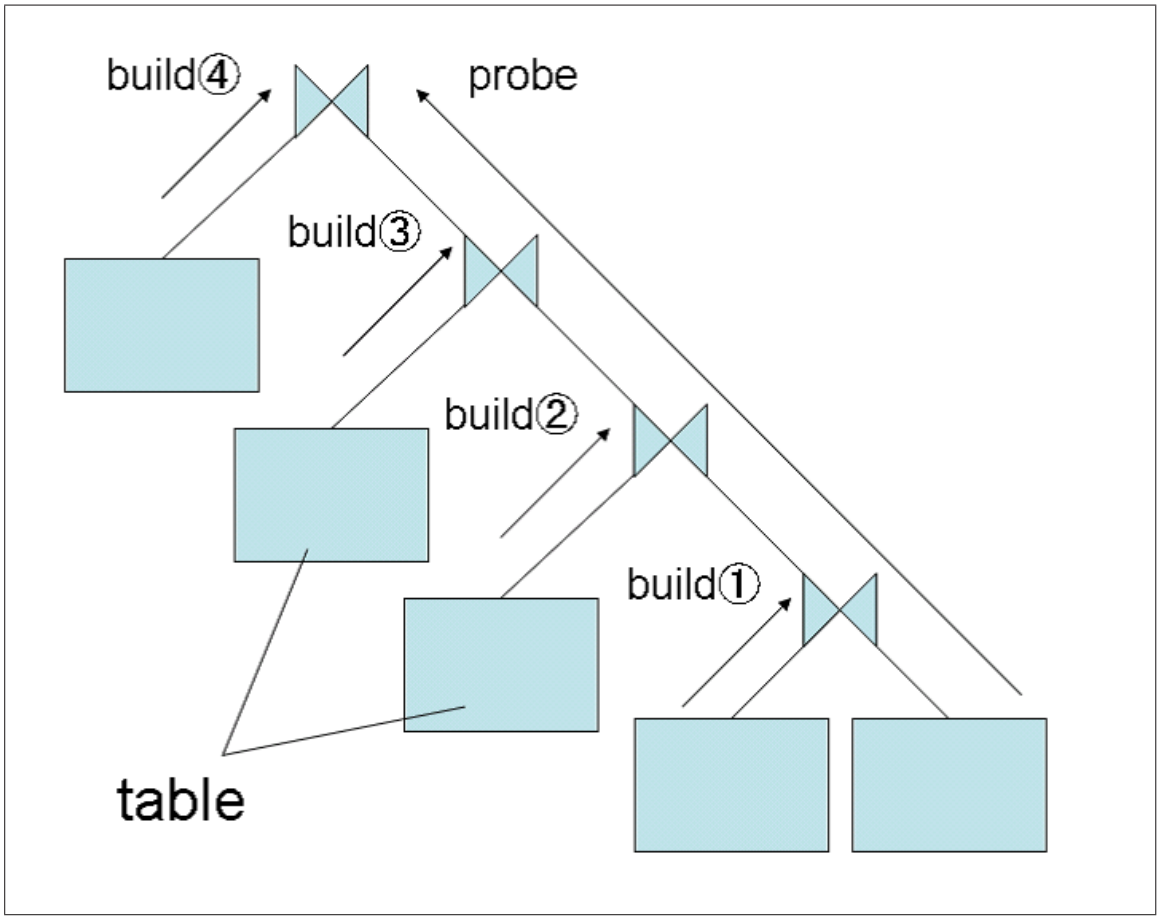

図 3.6: Hash Join (right deep)

逆に、図 3.7 のような形の実行プラン木を left deep と呼ぶ。これは、下側にある Hash Join から build、probe を順番に行っていくものである。

この場合、テーブルは木の左下から右側に向かった順番で読まれることになり、読まれたテーブルは、左下の一つを除いて全て probe のために用いられることになる。従って、パフォーマンスを最重要として考えるシステムにおいては、処理の軽い build をメインで行う right deep にすることが好ましく、また、Hash Join の際に Hit するタプルの数がなるべく少なくなるような順番で Join を行うことが求められる。

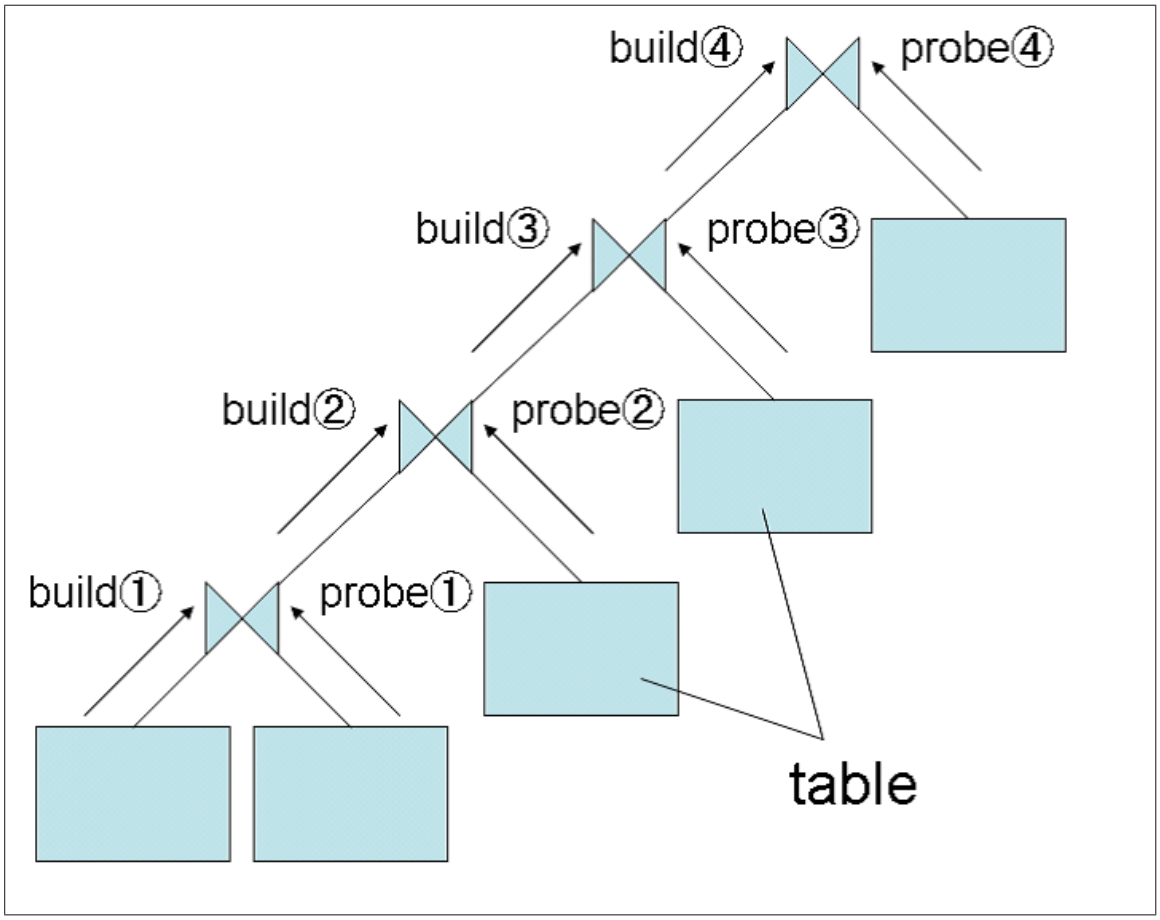

図 3.7: Hash Join (left deep)

3.5 問い合わせ実行計画

問い合わせ実行計画とは、トランザクションで送られてきた問い合わせに対して、どのような順序でデータベーステーブルの Join を行うか、また、どの種類の Join を行うか、どの index を用いるか、といった予定のことを表す。具体的には、図 3.6、図 3.7 にあるような木の、構造や、テーブルの順序、各 Join の種類、使用する index が決定されたものを意味する。

一般に、テーブルの Join は、抽出条件 (SQL の where 部) が同じであれば、どのような順序でどのような種類の Join を行っても、出力結果は同じになる。しかし、問い合わせ実行計画がうまく最適化されていないと、非常に時間がかかったり、ディスクアクセスが煩雑になったりする。よって、問い合わせやテーブルの特徴に応じて最適な問い合わせ実行計画を立てることは非常に重要であり、DBMS の性能を決定付ける。

また、問い合わせ実行計画を管理するのは DBMS であるが、これには非常に多様な種類が存在する。例えば、オープンソースの DMBS としては、PostgreSQL[12]、MySQL[11] などが挙げられるが、これらでは問い合わせ実行計画をユーザが自由に決定することはできない。商用の DBMS である HiRDB、Oracle などは、クエリ文を書き換えることにより、問い合わせ実行計画を自由に変更する機能を備えている。よって、本実験では HiRDB を用い、更に SQL 文を自ら書き換えることにより、問い合わせ実行計画をコントロールする。

第4章　問い合わせ実行計画を利用した省電力化方式

4.1　概要

従来のラップトップ用ディスクの研究を始め、データセンタにおけるディスクに関する先進的な研究においても、ディスクの電力を削減する手段としては、ディスクの回転数を低くしたり、停止させたりするのが最も単純かつ有効である [10]。本研究においてもそれは変わらないが、問題はいかにして回転数を変更するタイミングを制御するかということにある。

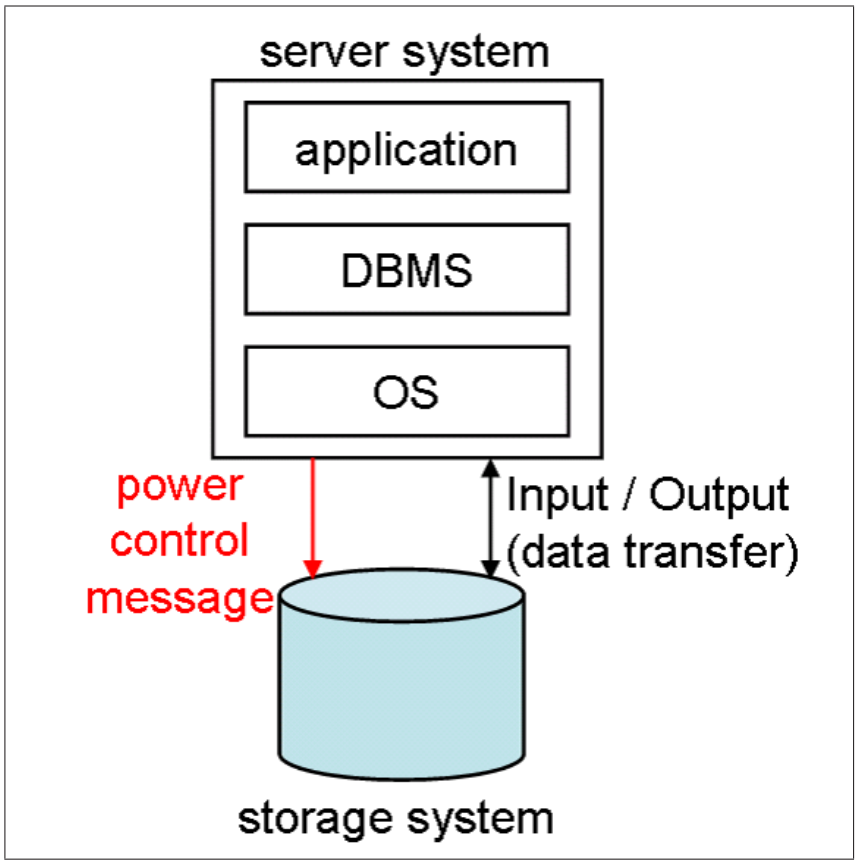

図 4.1: 電力制御機構を持つデータベースシステム概念図

図 4.1 に、本研究で想定するデータベースシステムの概念図を示す。赤い線で示されている、

電力モードを変更するメッセージを送れるようなインターフェースを加えることにより、以下のような考えを実現することができる。

従来の研究では、ディスクの idle 時間に応じて回転数を遷移させたり、各ディスクに対するアクセス頻度を学習してデータをディスク間で移動させるなど、ストレージ内部の情報をもとにディスクの動作を制御するしかなかった。しかし、本研究では、DBMS が自由に問い合わせ実行計画を決めることができることに着目し、その情報を元にディスクの動作を決定することを考える。つまり、データの場所が分かっていれば、動作する必要のあるディスクと必要のないディスク、またその時間などを把握することができ、アプリケーション、DBMS から能動的にディスクを制御することで、従来より効率のよい電力削減を図ることができる。

4.2 Join の種類の選択

3.3 節でも述べたように、Join にはいくつかの種類が存在する。

Nested Loop Join は、両テーブルの各タプルを全通り比較する方法で、単純だが計算量が多く、あまり効率が良くない。また、図 4.2[1]を見ると、縦軸がディスク上の位置、横軸が時間、赤い点が read アクセスを表しているが、ディスクアクセスがランダムになっていることが分かる。本研究では一つのディスクにアクセスしている間に、不要なディスクの回転数を抑えることが目的なので、このようなランダムアクセスは目的にそぐわない。

また、Sort and Marge Join は、Join の両テーブルを一度ずつシーケンシャルに読めばよいだけなので、先に説明した目的に合う形にはなるが、タプルがソートされているか読み込んだ後にソートすることが条件となるため、これにもかなりの計算量とメモリ量を必要とする。

一方、Hash Join は、片方のテーブルで Hash テーブルを作成し、片方のテーブルを読みながら、作成したテーブルにつき合わせるという形をとる。この場合、Hash テーブルを作成する方のテーブルをサイズの小さいものに、付き合わせる方のテーブルをサイズの大きいものにし、更に、早い段階の Join でのヒット率が小さい組み合わせを指定すれば、後の Join の計算量もメモリ量も抑えることができる。図 4.3 を見ると、ディスクアクセスもテーブルを一度ずつ、シーケンシャルに読めばよいということが分かる。従って、本研究の目的に最も適する方式となる。

[1]図 4.2、図 4.3 は、本研究室内で開発された、ディスク I/O 監視ツールにより生成されたものである。

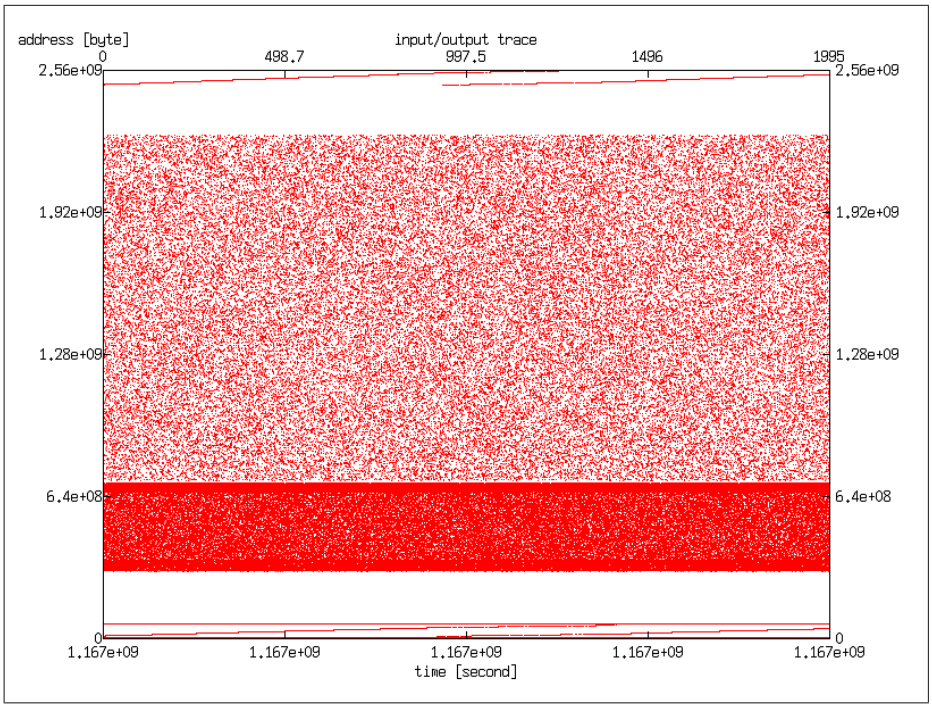

図 4.2: Nested Loop Join のアクセスパターン

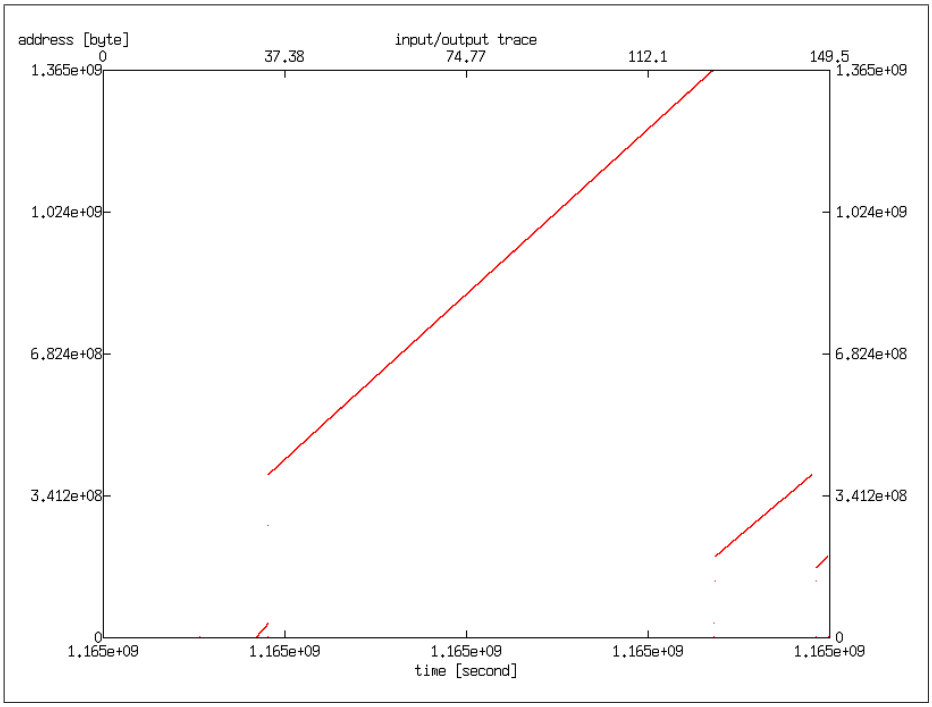

図 4.3: Hash Join のアクセスパターン

4.3　ディスクの状態遷移モデル

本研究では、ディスクを読み書き可能な回転数から低い回転数に遷移させることで電力削減を図る。本節では、ディスクの取りうる状態に関して説明を行う。

今回、データベースを保存するディスクとして、HGST の Deskstar T7K250[16] を使用した。その理由は、idle 時の電力を数段階変えることができるからである。これを例にして、ディスクの各状態を説明していく。図 4.4 を参照しながら、ディスクのそれぞれの状態に関して簡単に説明する。

・normal active とは、ディスクが read や write を行っている最中の状態であり、最も電力が高い。
・normal idle とは、スループットが 0MB/s であり、ディスクは回転したまま、読み書きも、電力削減に繋がる特殊な動作も行っていない状態を表す。
・low-power idle とは、normal idle の特殊型で、ディスクのヘッドをランプに乗せ、ヘッドの動作に関わる電力を削減している状態のことを表す。
・low-RPM standby とは、low-power idle の状態から更にディスクの回転数を落とすことにより、電力を削減している状態のことを表す。
・standby とは、low-RPM standby の回転数を 0round/min にしたものである。
・power off とは、ディスクに通電していない状態であり、消費電力は 0W となる。

本研究では、normal active、normal idle、low-RPM standby の三つの idle 状態に着目する。更に、それぞれの電力を測定し、モデルとして使用する。

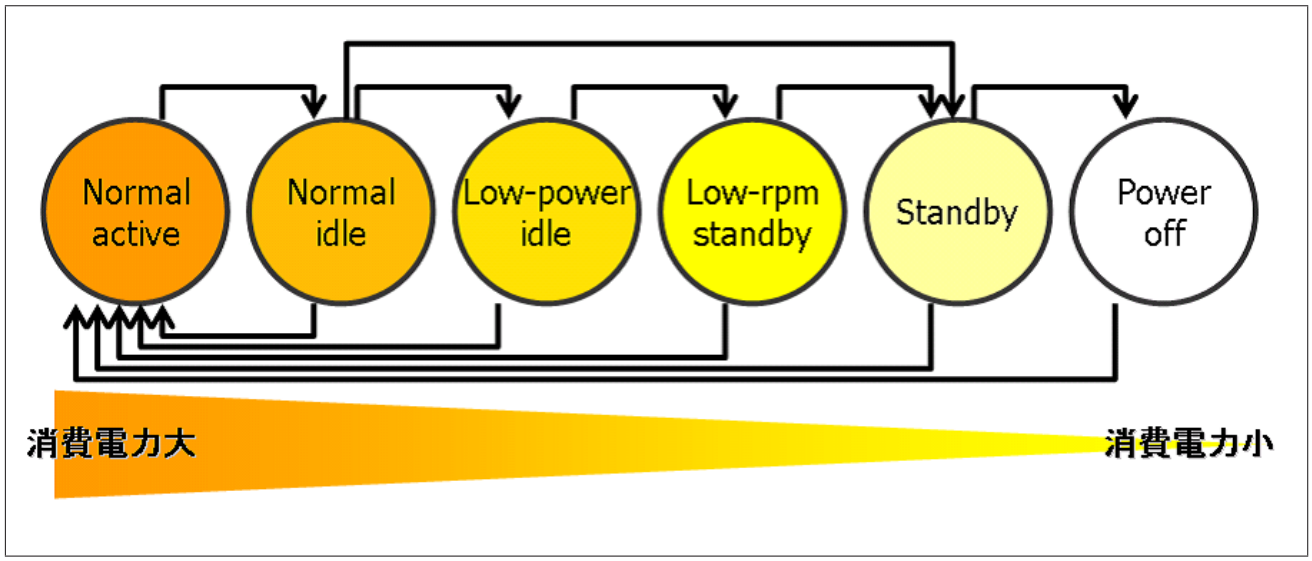

図 4.4: ディスクの取りうる状態と遷移

4.4 省電力化方式

図 4.5 を参照しながら、提案する方式の流れを説明する。

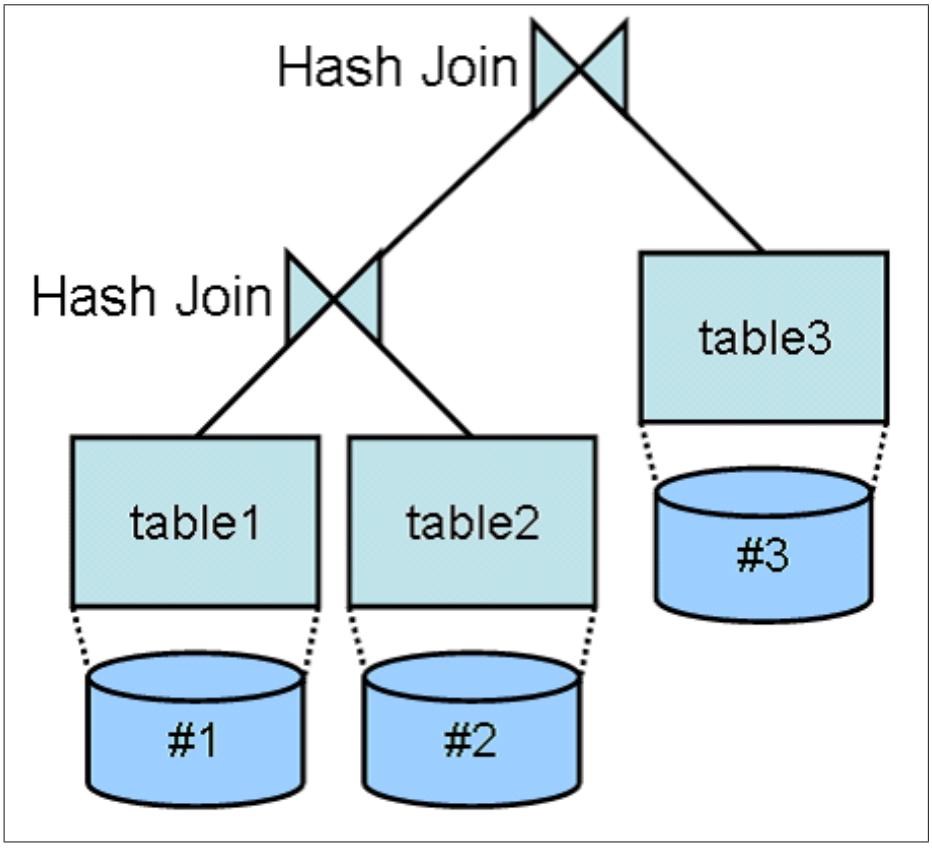

図 4.5: 想定するテーブル配置

今回の実験 (電力を測定した環境) では、全てのテーブルが一つのディスクに格納されている。しかし、本研究では、一つのテーブルを一つ (またはそれ以上) のディスクにシーケンシャルに割り当てられているような状況を想定する。例えば、table1、table2、table3 をそれぞれを異なるディスク#1、#2、#3 に格納しておく。これら三つのテーブルの Join を図 4.5 の問い合わせ実行計画に基づいて行うとする。
step1:table1 を build するためにディスク#1 を activate する。その間、ディスク#2、#3 を spin down し、table1 の build を待つ。
step2:table1 の build が完了したら、次にディスク#1 を spin down、ディスク#2 を activate して table2 による probe を行う。その間もディスク#3 は動作する必要がないため、probe と第 1Join 後のテーブルが build されるのを待つ。
step3:それらが完了したら、ディスク#2 を spin down、ディスク#3 を activate して、table3 の probe を行う。

ストレージに spin up/down を直接指示することができれば、このようにして、動作する必要

表 4.1: TPC-H のテーブルと実験ディスク上での位置 (Byte)

table name	start position	end position	table size
part	1359872	32964608	31604736
supplier	33341440	35028992	1687552
partsupp	35438592	166232064	130793472
customer	166526976	194232320	27705344
orders	194314240	390725632	196411392
lineitem	390938624	1364836352	973897728

のあるディスクのみを DBMS が定めた問い合わせ実行計画に基づいて特定することにより、より効率の良い省電力化方式が可能になる。

4.5 ベンチマーク TPC-H のテーブル構成の特徴を利用した問い合わせ実行計画

本研究では、ベンチマークとして TPC シリーズ [13] の TPC-H[14] を用いた。TPC-H は、8 つのデータベーステーブルと 22 の問い合わせから構成されている。

オンラインショップなど、小さな問い合わせが頻繁にやってくる OLTP を主体とした TPC-C とは特徴が異なり、TPC-H は幅広い産業分野で用いられる意思決定支援システム (DSS:Decision Support System) を主体としている。例えば、企業において、この商品の原価が 5 %上がった時の他方面への影響などをシミュレートしたりする。そのため、DSS は非常に膨大なデータを扱い、複雑な問い合わせを実行する。すなわち、一つの問い合わせに対する処理が非常に長時間かかることになる。本研究はその点に着目し、更に、一つの問い合わせに対しては、アプリケーションが定める問い合わせ実行計画に基づいて、非常に確定的な処理が行われることを利用して省電力を図る。

TPC-H のデータベースは、スケールファクタ (データサイズの尺度) が 1.0 の時、全体が約 1.3GB 程のサイズとなる。このデータベースを用いて TPC-H の各問い合わせを処理する時、一般的なデスクトップマシンを用いて、数百秒〜数千秒という単位の処理時間がかかる。このとき、主にディスク I/O がボトルネックとなっている場合が多い。

しかし、実際に産業分野の現場で用いられている DSS のデータベースは、TB〜PB というサイズのデータを有する。このことからも、DSS の問い合わせとその Join 演算がいかに膨大な作業であるか、そして、そのデータベースを保持するための電力コストがいかに高いものであるかが分かる。

TPC-H のデータベースは、表 4.1 に表されるテーブルから構成される[2]。表は、実験で用いた

[2] この他に、nation、region というテーブルも存在しているが、サイズが非常に小さいため省略する。実際には、part よりも内側のアドレス (ディスクの物理的な位置は外側) に位置している。

ディスク上での各テーブルの位置と、テーブルサイズも示している。加えて、スケールファクタは 1.0 である。

以上のような状況で、TPC-H のクエリ Q8 の実行を行う。

まず、TPC-H により得られる Q8 の SQL 文は以下のようになっている。

```
with all_nations(o_year, vlm, nation)
  as
   (select
      year(o_orderdate) as o_year,
      l_extendedprice * (1 - l_discount) as vlm,
      n2.n_name as nation
    from
      part, supplier, lineitem, orders, customer,
      nation n1, nation n2, region
    where
      p_partkey = l_partkey
      and s_suppkey = l_suppkey
      and l_orderkey = o_orderkey
      and o_custkey = c_custkey
      and c_nationkey = n1.n_nationkey
      and n1.n_regionkey = r_regionkey
      and r_name = 'AMERICA'
      and s_nationkey = n2.n_nationkey
      and o_orderdate between DATE('1995-01-01') and DATE('1996-12-31')
      and p_type = 'ECONOMY ANODIZED STEEL')
  select
  o_year,
  sum(case
    when nation = 'BRAZIL' then vlm
    else 0
  end) / sum(vlm) as mkt_share
  from all_nations
  group by o_year
  order by o_year
  without lock;
```

HiRDB の場合、これを、以下のような形で指定することにより、Hash Join の left deep として問い合わせ実行計画を立てる。

```
with all_nations(o_year, vlm, nation)
```

```
as
 (select
    year(o_orderdate) as o_year,
    l_extendedprice * (1 - l_discount) as vlm,
    n2.n_name as nation
  from
    region inner join /*>> by hash <<*/
    (nation n2 inner join /*>> by hash <<*/
    (nation n1 inner join /*>> by hash <<*/
    (customer inner join /*>> by hash <<*/
    (orders inner join /*>> by hash <<*/
    (supplier inner join /*>> by hash <<*/
    (lineitem inner join /*>> by hash <<*/
    part
    on p_partkey = l_partkey)
    on s_suppkey = l_suppkey)
    on l_orderkey = o_orderkey)
    on o_custkey = c_custkey)
    on c_nationkey = n1.n_nationkey)
    on s_nationkey = n2.n_nationkey)
    on n1.n_regionkey = r_regionkey
  where
    r_name = 'AMERICA'
    and o_orderdate between DATE('1995-01-01') and DATE('1996-12-31')
    and p_type = 'ECONOMY ANODIZED STEEL')
select
o_year,
sum(case
  when nation = 'BRAZIL' then vlm
  else 0
end) / sum(vlm) as mkt_share
from all_nations
group by o_year
order by o_year
without lock;
```

right deep を指定したい場合は、“`inner join /*>> by hash <<*/`” の前後にあるテーブルの順序を変えればよい。

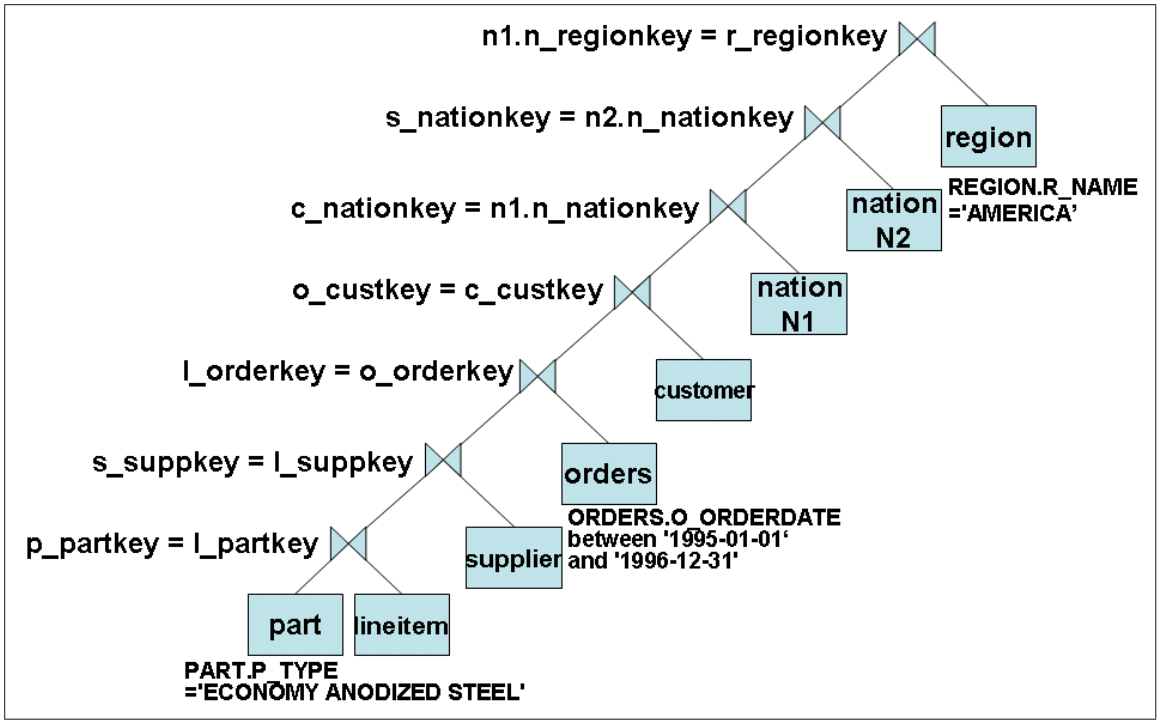

図 4.6: TPC-H Q8 left deep の問い合わせ実行計画

このようにして left deep を指定した形で実行を行うと、図 4.6 のような問い合わせ実行計画を得る。このとき、図 3.7 に則って処理の順番を分析すると、まず、テーブル part がディスクから読み込まれ、アトリビュート p_type が “ECONOMY ANODIZED STEEL” であるタプルのみが抽出され、build が行われる。これにより作成された Hash テーブルはメモリ上に展開されており、テーブル lineitem をディスクから読み込みながら、probe が行われる。このとき、テーブル part のアトリビュート p_partkey が、テーブル lineitem のアトリビュート l_partkey と一致するタプルのみが Join の結果出力されるテーブルのタプルとなる。このようにして作成されたテーブルは再び build され、今度は、テーブル supplier によって probe される。これを繰り返し、全ての Join が終了したとき、最終的に一つのテーブルが出力されることになる。

このときのディスクアクセス状況は、先に掲載した図 4.3 のような形になる。

また、問い合わせ全体の作業のうち、ディスクアクセスと大量のデータ処理を伴うこの Join が、処理時間のほとんどの割合を占めることになる。よって、この Join の段階で消費電力を抑えることは非常に有効な手段であるということができる。

第5章　ディスク消費電力のモデル化

5.1　実験環境

本節では、各種実験環境について説明する。

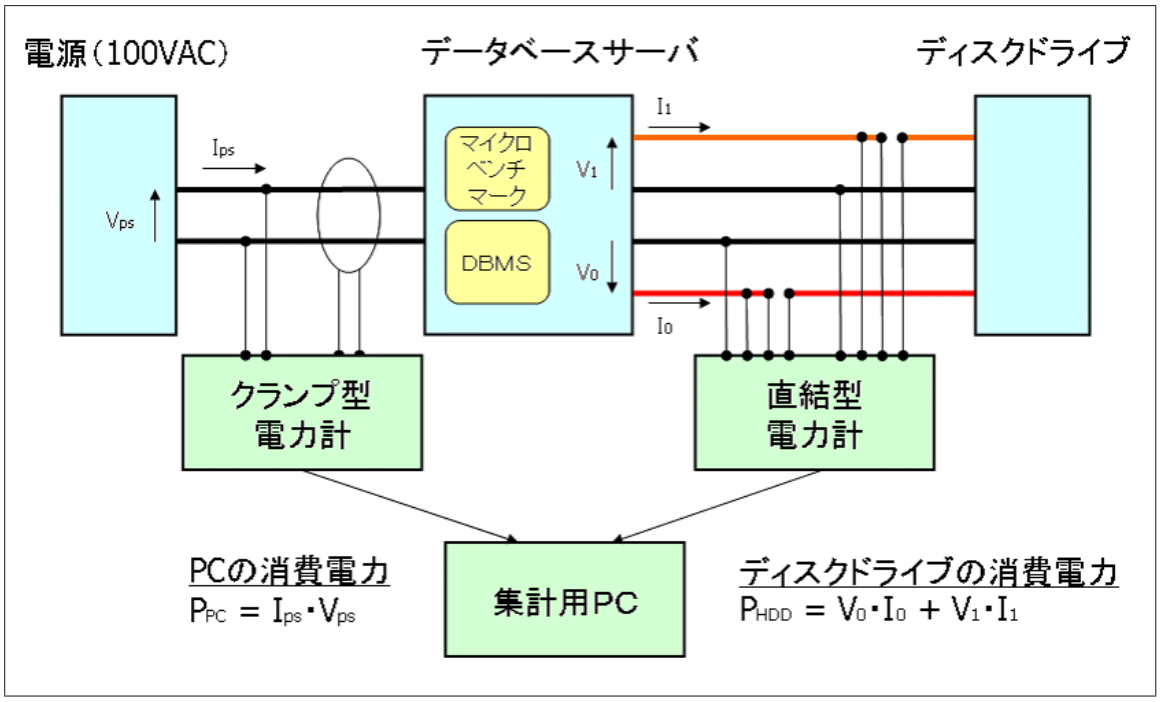

図 5.1: ディスク電力測定回路の概略

まず、図 5.1 は、ディスク電力を測定するための回路の概略図である。DB サーバからディスクへは 4pin の電源ケーブルを介して電力供給が行われ、赤線は 5V、黄線は 12V を表している。それらの線を YOKOGAWA 製の電力計、WT230 に通し、電流を測定する。また、赤線と黒線 (GND)、黄線と黒線の間にもクリップを噛ませ、電圧を測定する。この電流と電圧二つずつから得られた電力二つの合計が、ディスクの電力となる。更に、DB サーバ全体の電力を把握するために、YOKOGAWA 製の電力計、CW120 を用いて交流電流と電圧を監視している。

次に、DB サーバの状態を示す。DB サーバとしては Gateway 製デスクトップマシンを用い、CPU は Pentium4 1496.361MHz、メモリは 381508kByte である。OS は、Redhat enterprise LINUX Version 3 を用いている。HGST 製 T7K250 を IDE インターフェースのセカンダリ/マスターに

表 5.1: T7K250(HDT722516DLAT80 PATA) のスペック

シークタイム	8.5ms
回転数	7200rpm
キャッシュ	8MB
サイズ	160GB
ディスク枚数	2 枚
ヘッド	3 本
データ転送速度 (最大)	33MB/s
消費電力 (Idle 時)	5.2W

接続し、更に、raw キャラクタデバイス /dev/raw/raw1 を、ブロックデバイス /dev/hdc1 にバインドすることにより、raw デバイスとして取り扱っている。

最後に、実験用ディスク T7K250 のスペックを、表 5.1 に示す[1]。

5.2 active 状態におけるスループットベースのディスク消費電力モデル

本節では、4.3 節で説明した normal active 状態の電力に関して説明を行う。

normal active 状態では、read または write を行っている。更に、それらはシーケンシャルとランダムに分けることができる。シーケンシャルではヘッドの動きがほとんどないので電力は小さく、逆にランダムではヘッドに費やす電力が大きいと考えられる。

また、各 I/O に対してハッシングやその他の処理で次の I/O までに時間がかかる場合もある。この時、I/O 以外の処理によってスループットが変化する。このスループットの違いによってもディスクの電力が変わってくると考えられる。

以上を踏まえ、本研究では、独自の I/O 負荷生成ツールによって作られた、sequential read、random read の電力を測定する[2]。また、その際に、各 I/O の合間にアプリケーションの休止を挟むことにより、スループットの調節を行う。これにより、様々なスループットにおける電力を測ることが可能になる。更に、今回用いたツールでは、1I/O のサイズを自由に変えることができる。よって、4096Byte、16384Byte、65536Byte の三種類で実験を行う。

以上を行うことにより、図 4.4 で示された実現可能な状態全てのモデルが出来上がることになる。これにより得られた電力モデルに、ベンチマーク実行中のスループットと実行時間、idle 状態の時間を適用することによって、ディスク電力量を算出する。

[1] データ転送速度は、DB サーバのチップセットの制約でモード udma2 のため、比較的低い値となっている。

[2] 今回は、ベンチマークが read のみを使用するため、write は省略する。

5.2.1 シーケンシャルアクセス時の消費電力モデル

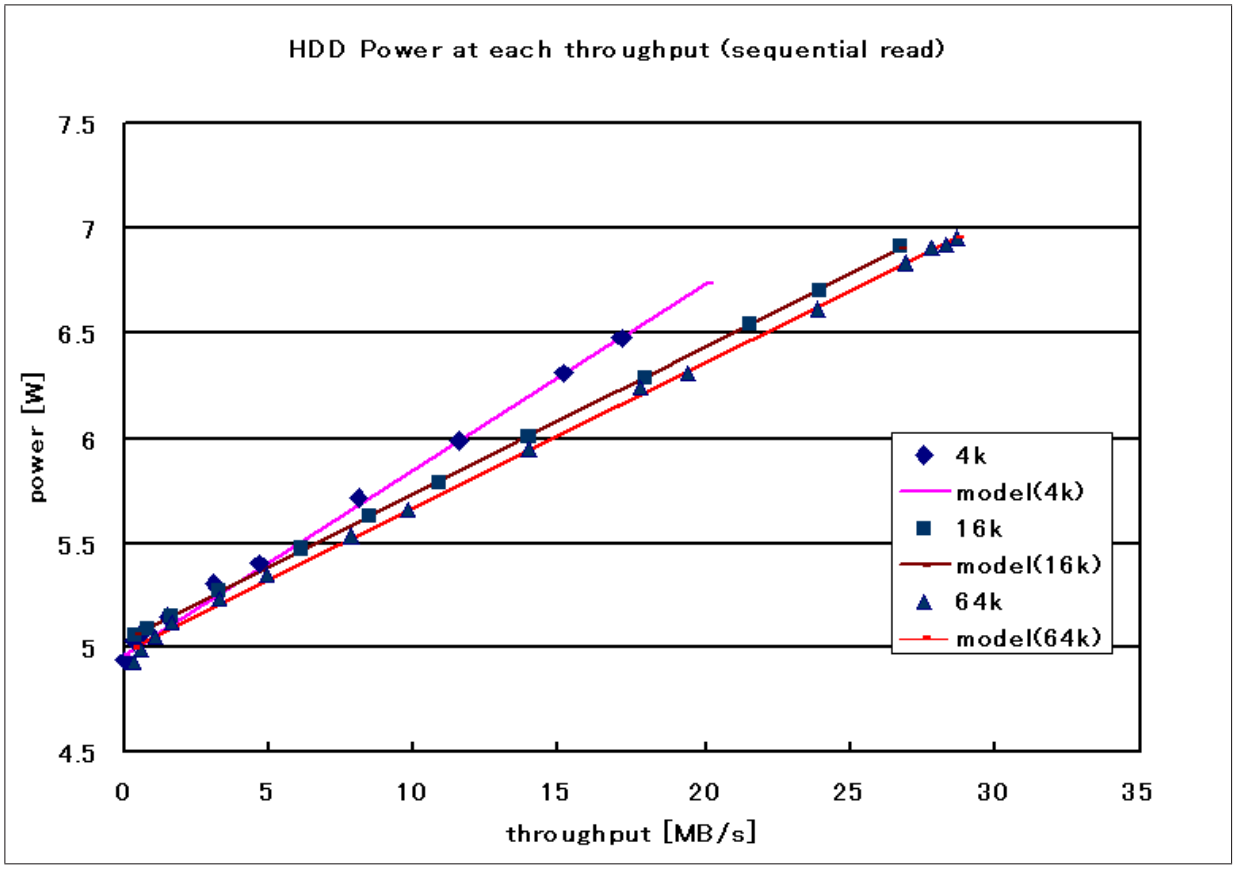

図 5.2: sequential read におけるスループットに対するディスク電力測定値とモデル

図 5.2 は、I/O 負荷生成ツールにより作られた sequential read における、スループットに対するディスク消費電力の測定値と、それに最小二乗法を適用することにより求めた電力モデルを図示したものである。かなり正確な直線になっており、また、1I/O のサイズが大きいほど電力効率が良い (傾きが小さい) ことが分かる。それぞれのデータに最小二乗法を適用してみると、得られた直線の式は、以下のようになった。

$Y = 0.0849X + 5.0176 \quad (4k)$
$Y = 0.0701X + 5.0324 \quad (16k)$
$Y = 0.0686X + 4.9859 \quad (64k)$
(X:スループット [MB/s]　Y:消費電力 [W])

5.2.2 ランダムアクセス時の消費電力モデル

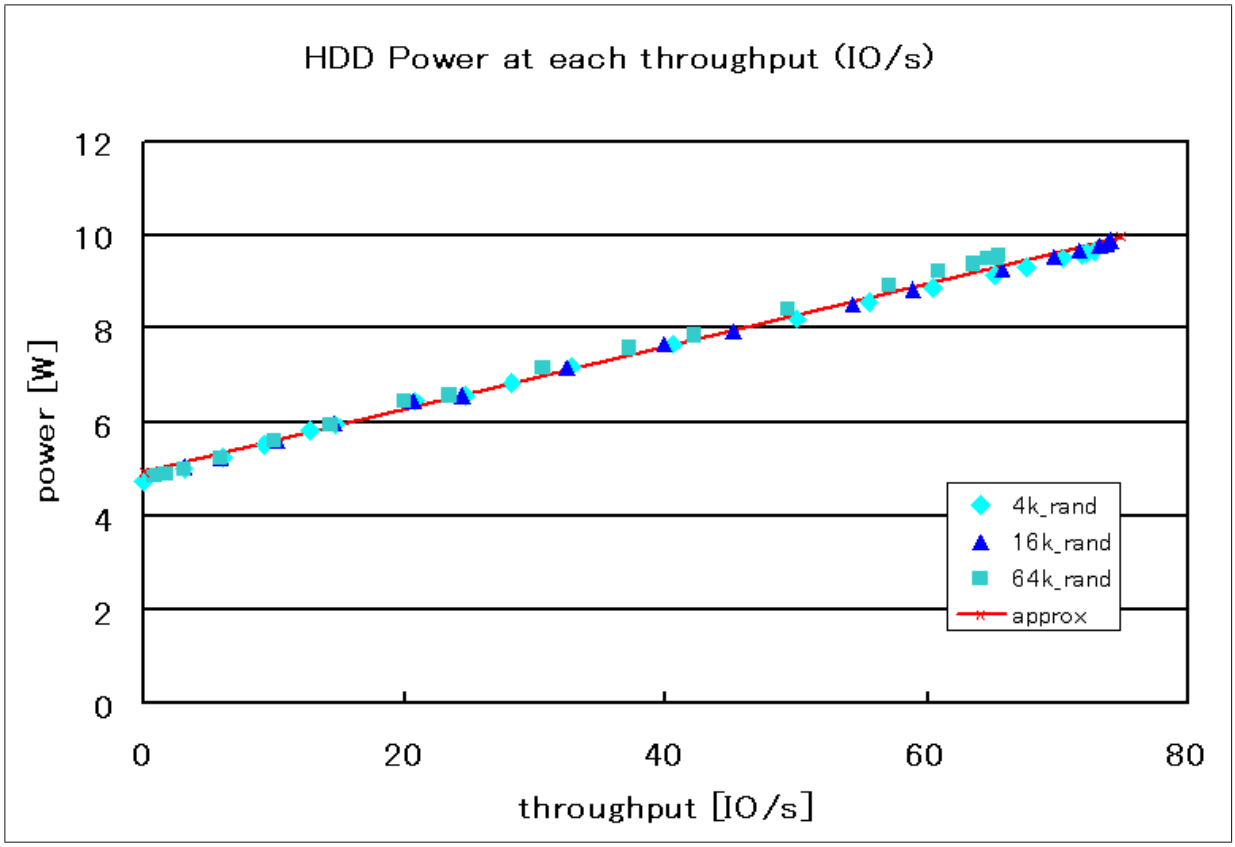

図 5.3: random read におけるスループットに対するディスク電力測定値とモデル

図 5.3 は、I/O 負荷生成ツールにより作られた random read における、スループットに対するディスク消費電力の測定値と、それに最小二乗法を適用することにより求めた電力モデルを図示したものである。横軸は、IO/s である。

こちらも sequential read 同様、ほぼ正確な直線となっているが、どの I/O サイズにおいてもほぼ同じ直線となっていることが分かる。これは、シーク時間がボトルネックとなって、ヘッドの動きの多寡が、I/O サイズよりも IO/s に依存しているためと考えられる。これら全てのデータに最小二乗法を適用してみると、得られた直線の式は、以下のようになった。

$Y = 0.0673X + 4.9177$

(X:スループット [IO/s]　Y:消費電力 [W])

5.3 idle状態におけるディスク消費電力モデル

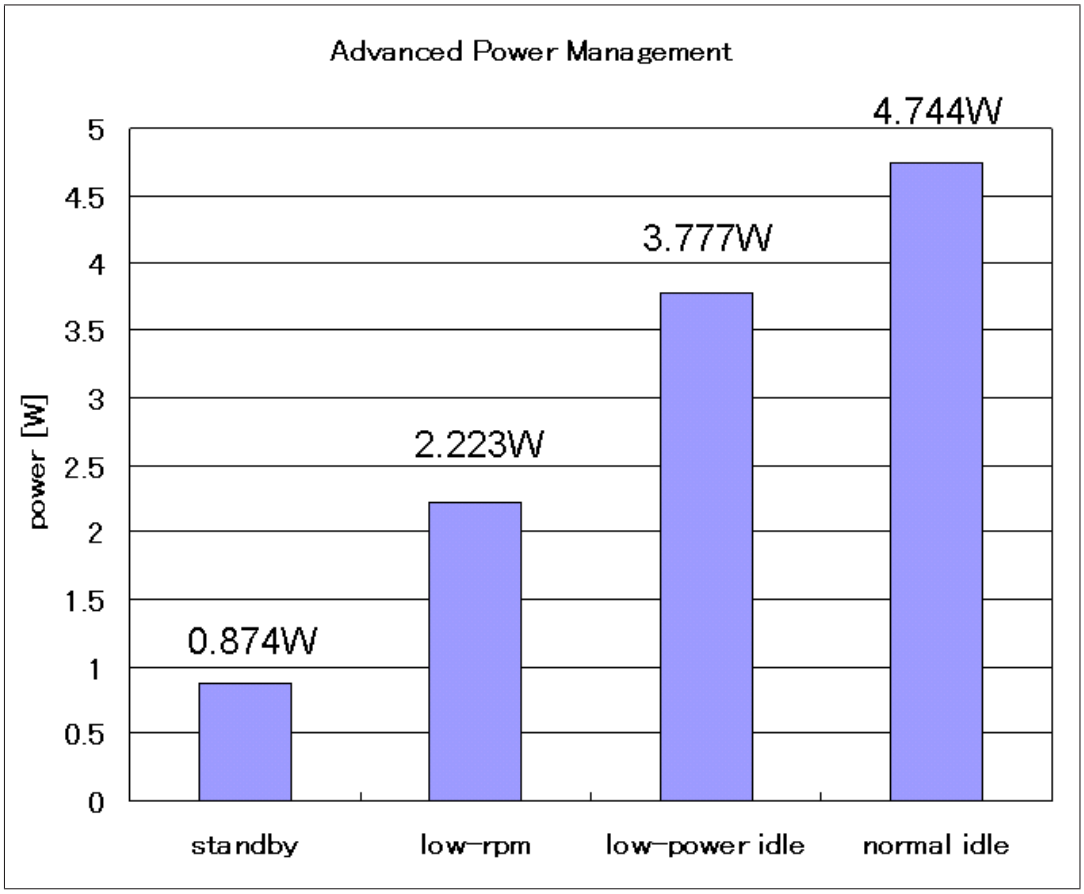

図 5.4: 各 idle 状態における消費電力

本節では、idle 状態における電力のモデル化を行う。4.3 節で説明した、normal idle、low-power idle、low-RPM standby、standby 時の電力を測定し、idle 時の電力モデルとして扱う。idle 時の電力なので、これにはスループットは関係なく、固定の値を定める。

図 5.4 は、normal idle、low-power idle、low-RPM standby、standby 状態における、各消費電力の実測値を示している。

今回の実験では、通常の idle 状態として normal idle を、省電力状態として low-RPM standby をモデルとして採用することにする。

表 5.2: 各省電力状態の、normal idle に対する電力削減率

状態	削減率 (実測)	削減率 (仕様)	復帰時間 (実測)	復帰時間 (仕様)
low-power idle	20.38 %	24 %	0.6sec	0.7sec
low-RPM standby	53.14 %	51 %	3.2sec	7.0sec
standby	81.58 %	89 %	7.2sec	15.0sec

また、表 5.2 は、low-power idle、low-RPM standby、standby 状態の、normal idle 状態に対する電力削減率の実測値と、white paper の値を示したものである。実測値は white paper で示されたものと概ね近い値となっている。

5.4 状態遷移時コストの消費電力モデル

最後に、4.4 節で紹介した各電力モード間の状態遷移コストをモデル化する。

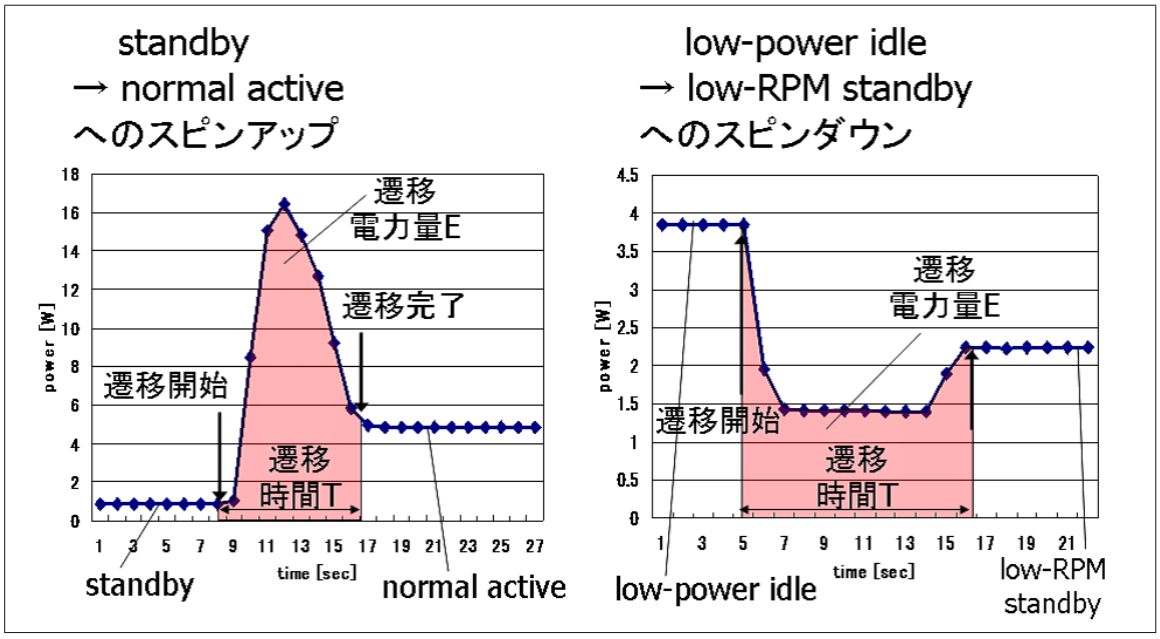

図 5.5: 状態遷移時の電力推移の例

図 5.5 は、standby から normal active への spin up と、low-power idle から low-RPM standby への spin down の状態遷移時の電力を 1 秒ごとにプロットしたものである。色で表している面積が遷移電力量であり、その幅が遷移時間である。また、spin up に関しては、I/O リクエストを一つ与えた時点から I/O がコミットされるまでの時間を遷移時間とし、spin down に関しては、電力が低下し始めてから安定するまでの時間を遷移時間とする。spin down の始点と終点に関し

ては、1 秒精度の電力グラフに頼らざるを得ないので、遷移時間が非常に短いものは、やむを得ず仕様上の値を借りることにする。

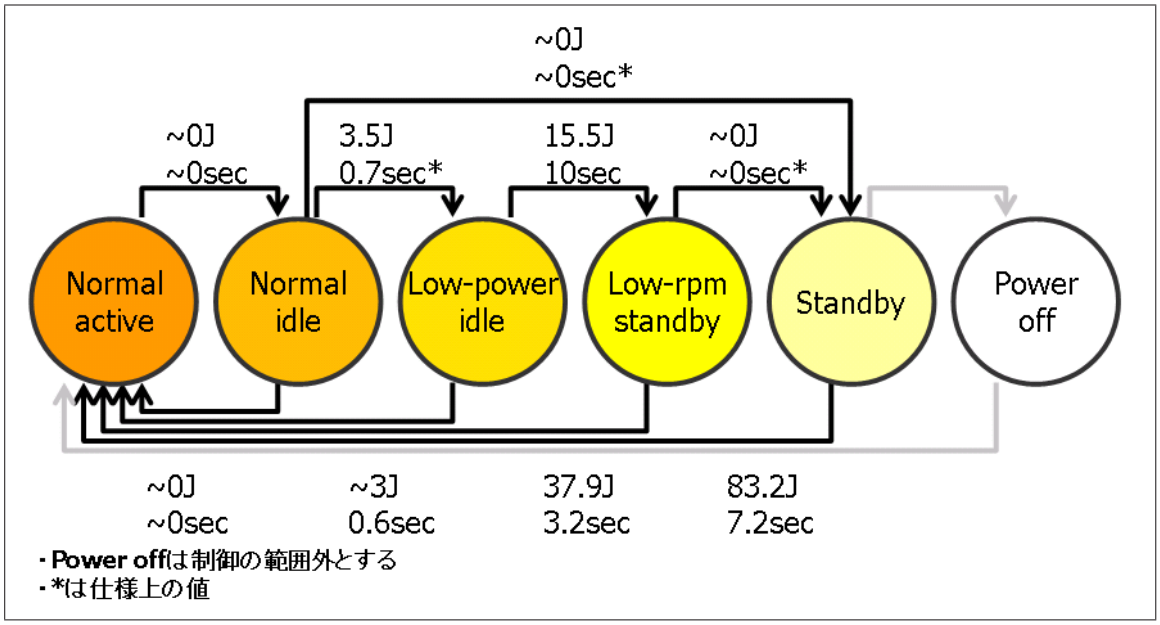

図 5.6: 状態遷移コスト

図 5.6 は、各状態遷移コストを表したものである。standby への spin down に関しては、状態遷移後に回転数を維持する必要がないため、スピンドルモータへの電力供給を止めることで遷移完了となり、1msec 以下の短い遷移時間となっている。

それぞれの状態の電力削減率を考慮すると、一度の spin up の電力的コストを打ち消すために各状態が取らなくてはならない状態継続時間は、low-power idle は約 3 秒、low-RPM standby は約 16 秒、standby は約 22 秒である。数百秒〜数千秒という処理時間の問い合わせにおいてこの値は比較的簡単に実現できる値であり、また、0.6〜7.2 秒という追加時間も全処理時間のわずか数%であるため、本研究が提案する手法の有効性を損ねない追加コストであるということができる。

第6章　評価と考察

本章では、モデル化したディスクの消費電力の妥当性を評価し、その後、NC、TPM、PACという三つのディスク制御方式の評価を行う。具体的には、Q8 left deep、Q8 right deep、Q9 left deep、Q9 right deepの四つに関して、上記三つの方式を想定し、各問い合わせ中の各フェーズ[1]にかかる電力量をモデルを用いて計算した後、全てのフェーズの電力量合計値を比較する。

まず、モデルを評価した後、実験に際して課した細かな前提条件を説明し、各問い合わせの計算結果 (電力量) を順に示していく。最後に全結果のまとめを行う。

6.1　ディスクの消費電力モデルの評価

まず、5.2節で作成した消費電力モデルを評価する。実際のベンチマークTPC-Hを、HITACH製のDBMS HiRDBを用いて実行した時のディスクにモデルを適用し、妥当性の評価を行う。具体的には、各テーブルの読み込みの際、シーケンシャルモデル、ランダムモデルを適用し、スループットと読み込み時間から各フェーズの電力量を計算する。それを全てのフェーズで行い、更に、読み込み以外の時間をidleのモデルに適用して電力量を合計する。こうして計算した値と、実際に問い合わせを実行している時間における電力量の実測値とを比較する。

[1]テーブル単位のbuild、またはprobeの一連の処理を指す。Q8、Q9の場合、一つの問い合わせは大小あわせて6〜8個のフェーズからなり、Hash Joinの場合は読み込むテーブルの数と等しくなる。

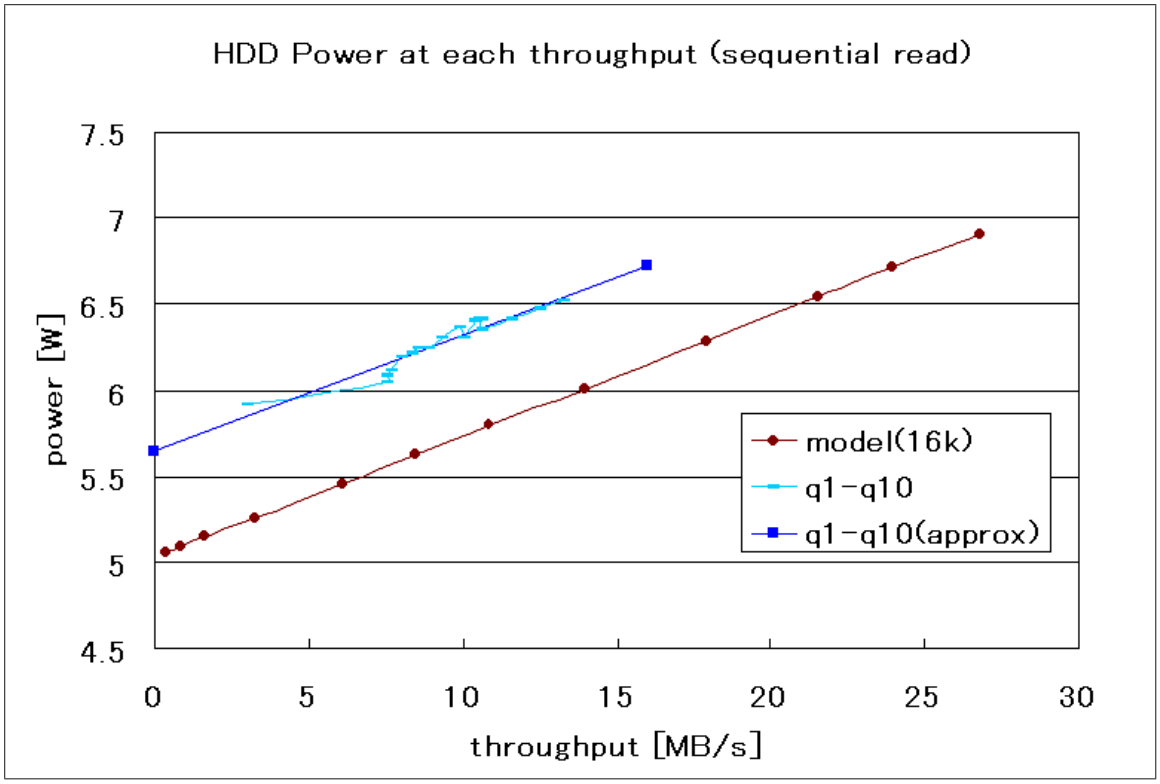

図 6.1: モデルによる電力と HiRDB における問い合わせ実行時の測定値のグラフ

図 6.1 は、TPCH Q1〜Q10 実行時の実測電力と、更にそれらに最小二乗法を適用して直線を得たグラフである。これを見ると、直線の傾きはほぼ 16k のモデルと一致し[2]、切片だけが 0.6W 程度上乗せされた形となっている。

つまり、モデル構築のために用いた独自の I/O 負荷生成ツールと、問い合わせによる I/O 負荷に何らかの違いがあるものと思われる。この原因に関しては、次頁で考察する。

[2]それでも若干 64k のモデルに近い傾きとなっているが、二つはほとんど変わりはない。

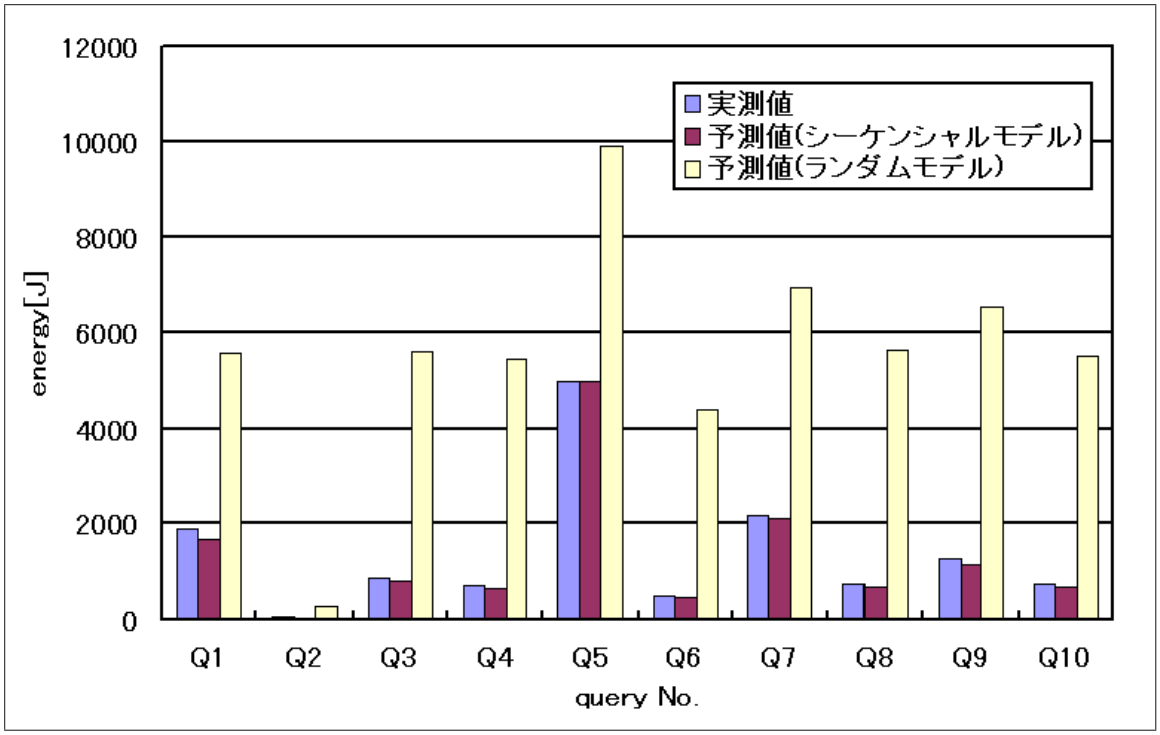

図 6.2: HiRDB における各問い合わせのモデルによる予想消費電力量と実測値

図 6.2 は、HiRDB における各問い合わせのモデルによる予想消費電力量と実測値とを比較したものである。また、Q2 など実行時間が短かったものを見やすくするために消費電力量を正規化したグラフが図 6.3 である。ランダムモデルによる予測値は大幅にずれているが、シーケンシャルモデルによる予測値の誤差は最大で 17 % (Q2) となっており、他の問い合わせも概ね 10 %未満に収まっている。これは、HiRDB による Join では、Nested Loop Join などランダムアクセスを行うような Join を避け、シーケンシャルアクセスである Hash Join を主に行うように最適化されるためである。

また、どの問い合わせでもモデルによる値より実測値の方が上回っているが、主な要因として考えられるのは、コントローラ系統やバッファキャッシュの消費電力である。I/O 負荷生成ツールでは、ディスクの一番初めのアドレスから 16kByte ずつ、シーケンシャルに read を行うのみであったが、問い合わせの実行においては、テーブルの read の他に、各テーブルの位置の指定や、ごくわずかの write I/O も行っている。従って、モデルを構築した時の動作よりも、問い合わせ実行時の動作の方が、コントローラや write バッファキャッシュが余分に働いている可能性がある。

これらを考慮に入れることで、より正確なモデル化に繋がる。この原因を調査し、更に正確なモデルを構築することは今後の課題としたい。

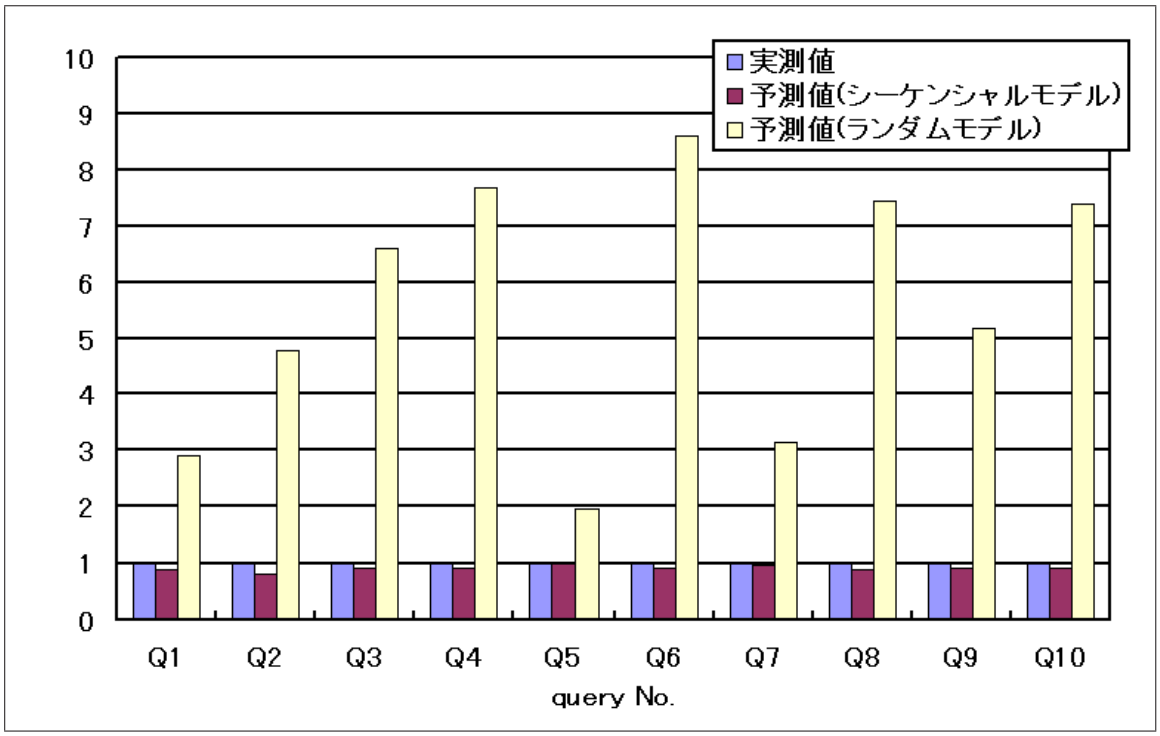

図 6.3: HiRDB における各問い合わせのモデルによる予想消費電力量と実測値 (正規化)

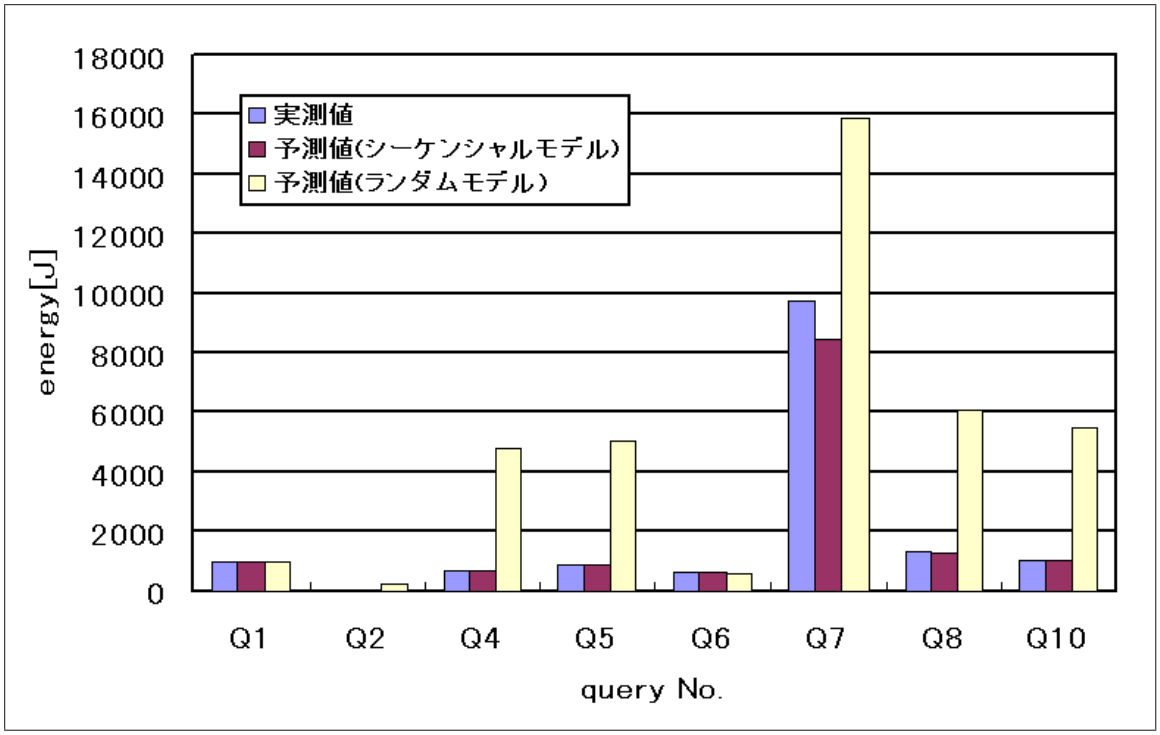

図 6.4: MySQL における各問い合わせのモデルによる予想消費電力量と実測値

更に、HiRDB だけでなく、MySQL に関しても同様の実験を行った。図 6.4 は、MySQL における各問い合わせ実行時の予想消費電力量の実測値と、モデルから算出した値を比較したものである。更に、消費電力量を正規化したグラフが図 6.5 である。ほとんどの問い合わせに関しては、HiRDB に近い、ほぼシーケンシャルなディスクアクセスであったため、シーケンシャルモデルがほぼ正確な値となっていた。実測値との誤差は、1 %から 6 %と非常に小さな値となった。

しかし、Q2、Q7 に関する実験結果は、ヘッドが頻繁に移動するランダムアクセスに近かったため、シーケンシャルモデルの予測値が他の問い合わせと比較すると大きくずれるものになっていたが、ランダムモデルによる予測値も正確であるとは言えなかった。

原因として考えられるのは、1I/O のサイズが大部分が 16kB であるが 1MB 以下の大きさも混じっていること、また、IO/s の値が特定しづらかったことなどが挙げられる。さらに、バッファキャッシュの影響で、実際にヘッドの動作とディスク読み取りの動作に関わっていない I/O が多数存在したため、IO/s の値が大きくなり、予測電力量が実測値より大きくなってしまったということも考えられる。

以上より、1I/O のサイズがまばらであるような問い合わせ、実行時間が非常に短いような問い合わせ、シーケンシャルとランダムの中間的な度合いのアクセスを行う問い合わせ、バッファキャッシュが有効に働く問い合わせなどは、このモデルを適用しづらい例であるといえる。これらを克服することも今後の課題として挙げられる。本論文後半の、モデルを用いた解析では、HiRDB を用いたブロックサイズ 16KB のシーケンシャルアクセスがほとんどであるため、シーケンシャルモデルが有効であると考えられるが、ランダムモデルに関してはまだ検討の余地が

ある。

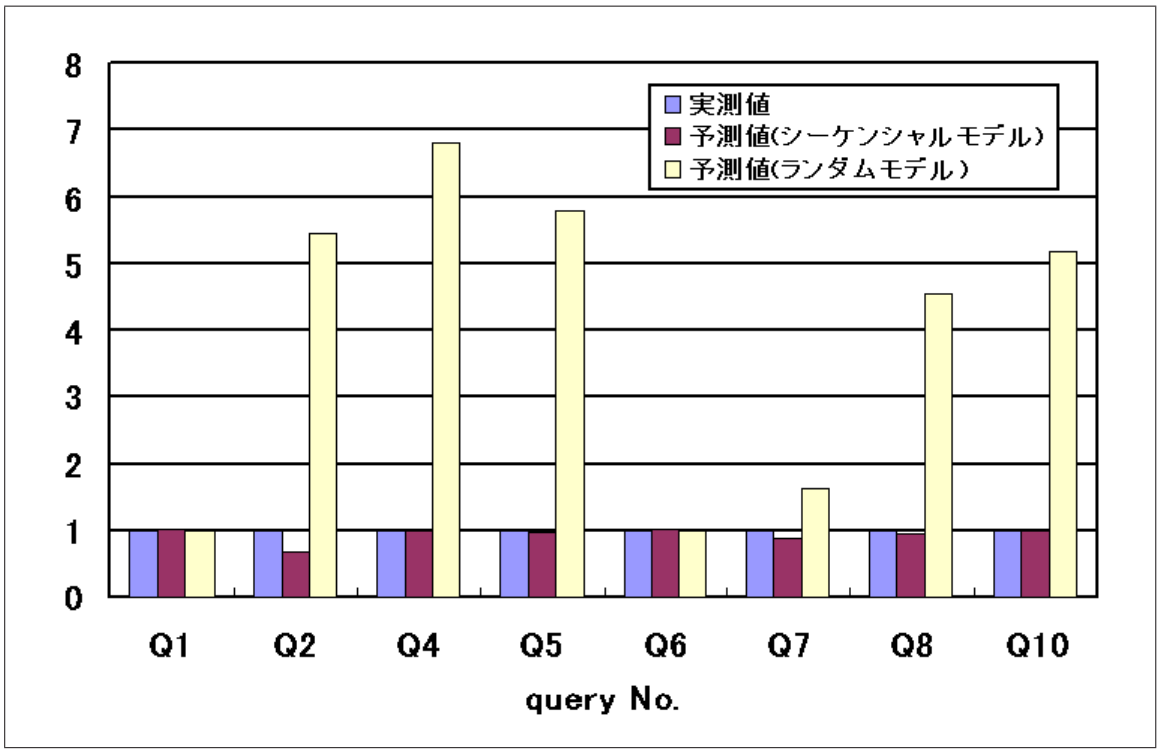

図 6.5: MySQL における各問い合わせのモデルによる予想消費電力量と実測値 (正規化)

6.2 実験の補足条件

以下のような条件で実験を行う。

・ベンチマークとして TPC-H を用いる。
・三つのディスクに、lineitem テーブル、orders テーブル、その他のテーブルがそれぞれ保存されている状態を想定し、ディスク 1、ディスク 2、ディスク 3 と呼ぶ。
・想定する状態としては、normal active、normal idle、省電力モードの三つを考え、それぞれを遷移するものとする。
・省電力モードとして、low-RPM standby と standby の二種類を考える。・spin up/down は、図 5.6 を元に、時間的、電力的コストを含めた値を算出する。・TPM(後述) の時間閾値は、180 秒、60 秒、30 秒とする。
・TPM、PAC では、初期状態は両電力モードではなく、normal idle であるものとする。つまり、PAC で初めから省電力モードに入るときは、一度 spin down する。
・小さなテーブルを対象とした細かい read のための spin up は無視する。

・それらの細かい read 部や idle 時間は非常に短いので無視する。ただし、Q9 の idle 時間は比較的長いので、全て idle として加算する。

6.3 Q8 left deep

まず、Q8 left deep の問い合わせ実行計画を再度確認するが、第 4 章の図 4.6 で例に挙げた通りである。これを元にして問い合わせを実行すると、図 6.6 のようなディスクアクセスを行う。縦軸はディスク上の位置、横軸は時間である。

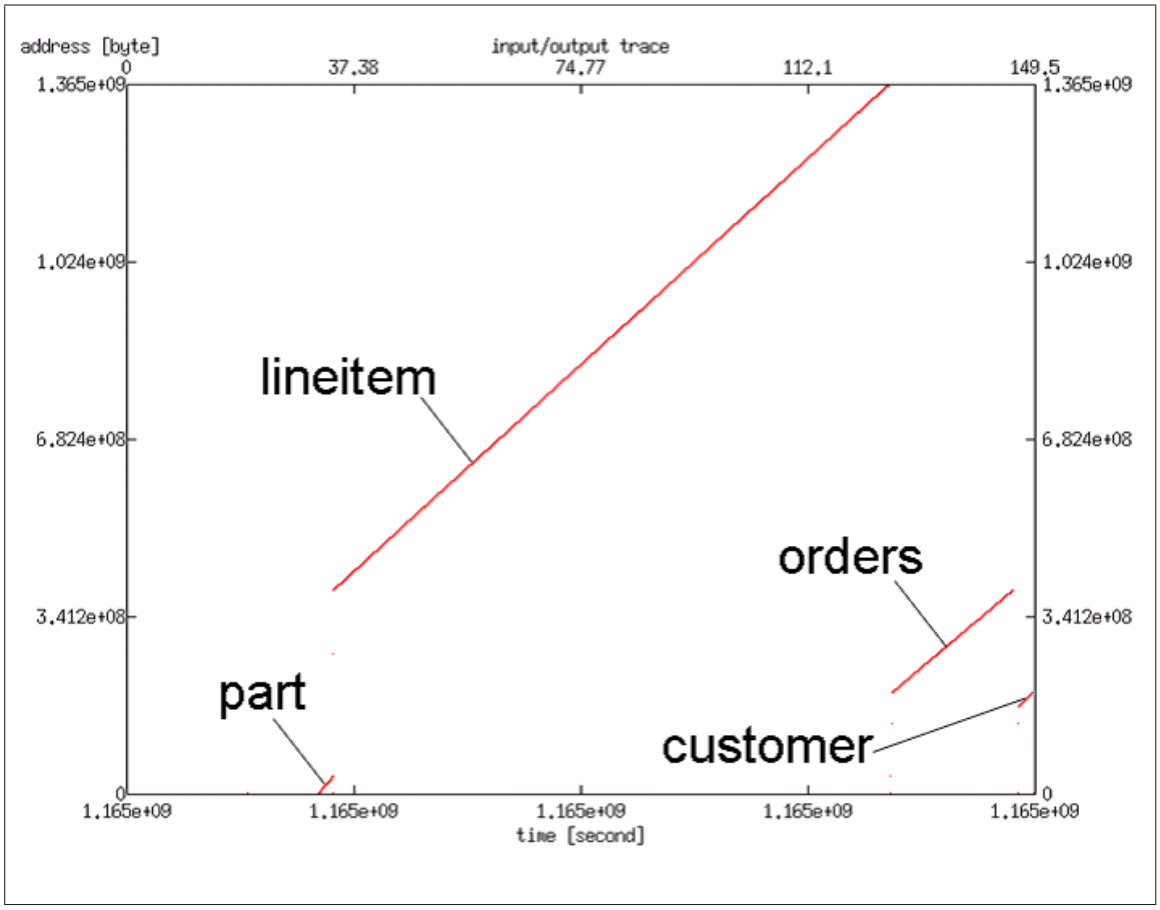

図 6.6: Q8 left deep のディスクアクセス

これを、4.5 節の表 4.1 と照らし合わせると、それぞれの直線がどのテーブルをあらわしているかが分かる。まず part テーブルを読み、build を行っている。次に lineitem テーブルを読み、probe を行っている。この後、part と lineitem の Join 結果でできたテーブルを build しているが、ほとんど間を空けずに orders の読み込みが確認できる[3]。これも probe を行うための read である。以下、customer が続く。

[3]問い合わせ実行計画を確認すると、lineitem と orders の間に supplier を読み、また、customer の後に nation1、nation2、region を読んでいるが、一瞬で終わっているため省略している。

表 6.1: Q8 left deep の各フェーズにおける電力量

table	type	time [s]	size [MB]	throughput	power [W]	energy [J]
part	build	2.416	31.605	13.082	5.949	14.374
lineitem	probe	91.658	973.898	10.625	5.777	529.528
orders	probe	19.821	196.411	9.909	5.727	113.515
customer	probe	2.462	27.705	11.253	5.821	14.332
others	others	1.535				
total		117.892	1229.619			671.749

この四つのフェーズを主な電力消費部として、一つのディスクからなる実環境からスループットを得て、それらをシーケンシャルモデルに適用することにより予想電力を計算すると、表 6.1 のような結果が得られた。

以上のような状況を踏まえ、ディスク制御方式三つの説明を行う。一つは、ディスクが idle であるときに何も省電力制御を行わない方式 (NC:No Control) であり、これを基準として他方式を比較する。もう一つは、ディスクの idle 時間が一定閾値を超えたときに自動的にディスクが省電力モードに移行するような方式 (TPM:Traditional Power Management) であり、これは図 4.1 の赤線で示したような新しい機構を加えることなく実現できる方式である。最後は、DBMS がディスクの idle 期間を把握し、能動的にディスクを制御する方式 (PAC:Proactive Control) であり、最も省電力が期待できる方式である。

表 6.1 を元に、NC、TPM、PAC におけるそれぞれのディスクの電力を、状態遷移コストも含めて計算すると、図 6.7、図 6.8 のようになる。それぞれ、省電力モードとして low-RPM standby、standby を用いている。

これら三つの方式における各ディスクの動作状況を例として表すと、図 6.9、図 6.10、図 6.11 のようになる。また、このときの省電力モードは low-RPM standby である。TPM では、ディスクアクセス後 30 秒間は normal idle となっており、PAC は NC の normal idle 部をアプリケーション側から能動的に low-RPM standby にしている様子が図示されている。また、状態遷移に伴って、少しずつ実行時間が長くなっていることも分かる。

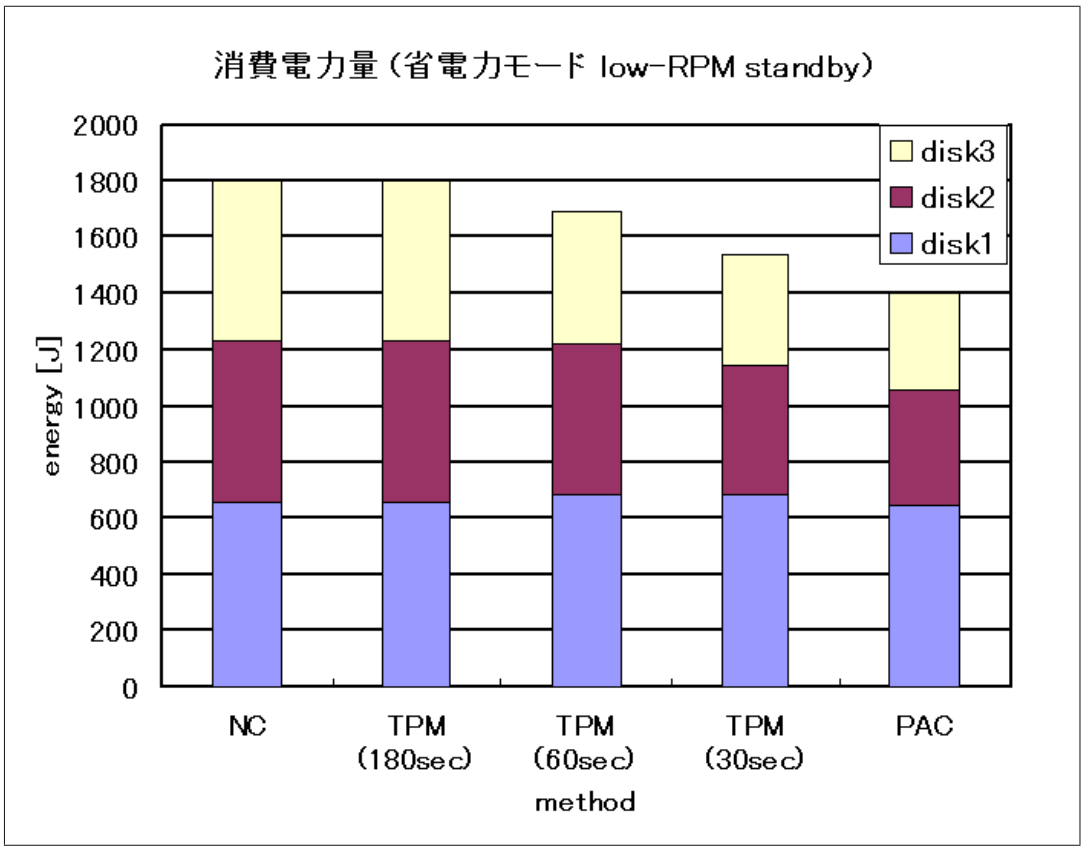

図 6.7: Q8 left deep における各制御方式の消費電力量 (省電力モード low-RPM standby)

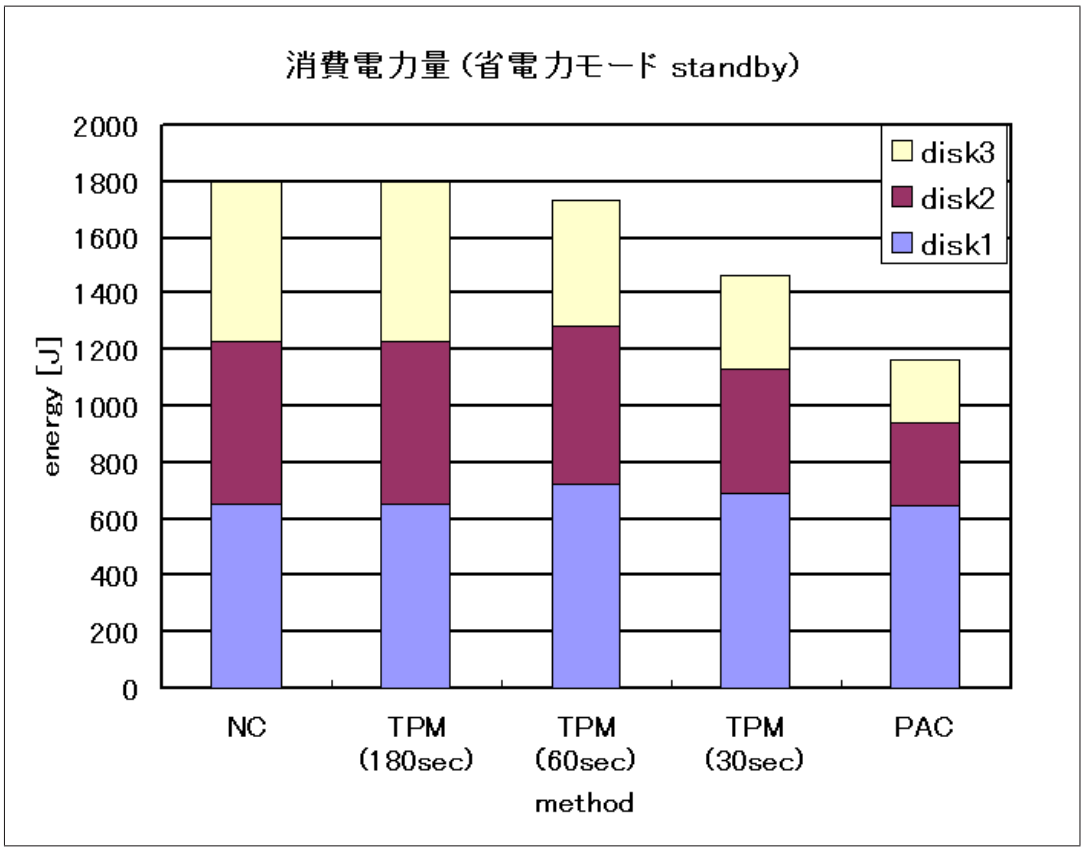

図 6.8: Q8 left deep における各制御方式の消費電力量 (省電力モード standby)

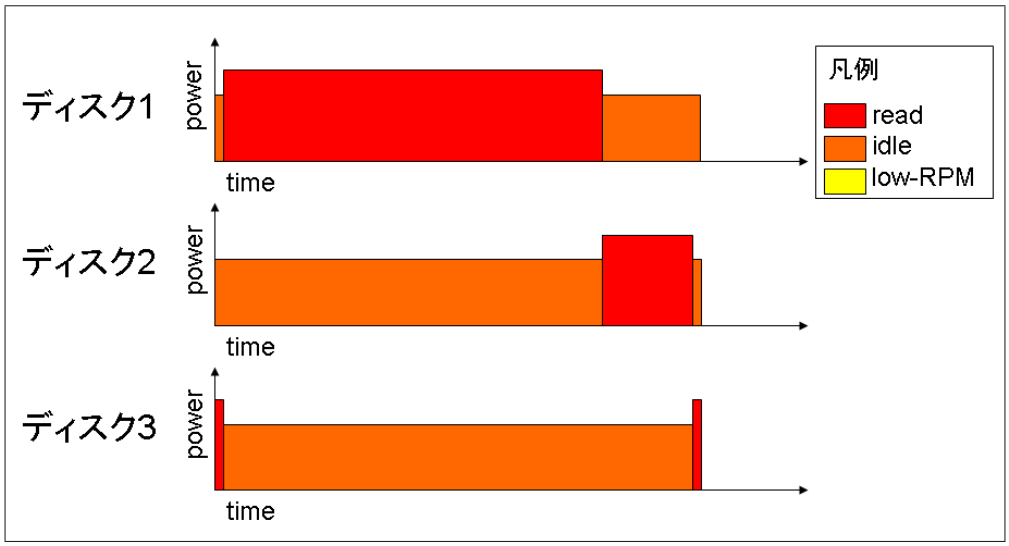

図 6.9: Q8 left deep NC における各ディスクの動作状況

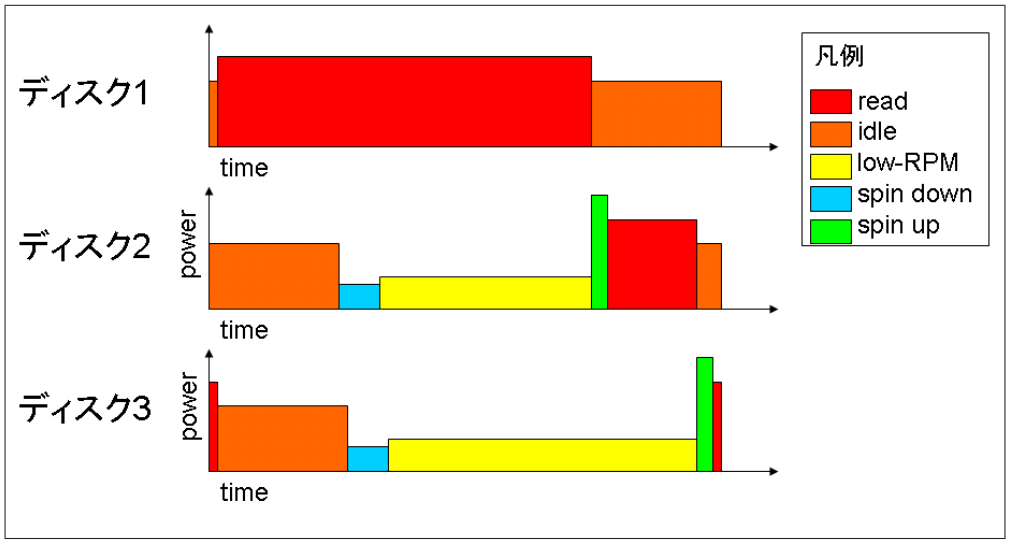

図 6.10: Q8 left deep TPM(閾値 30 秒) における各ディスクの動作状況

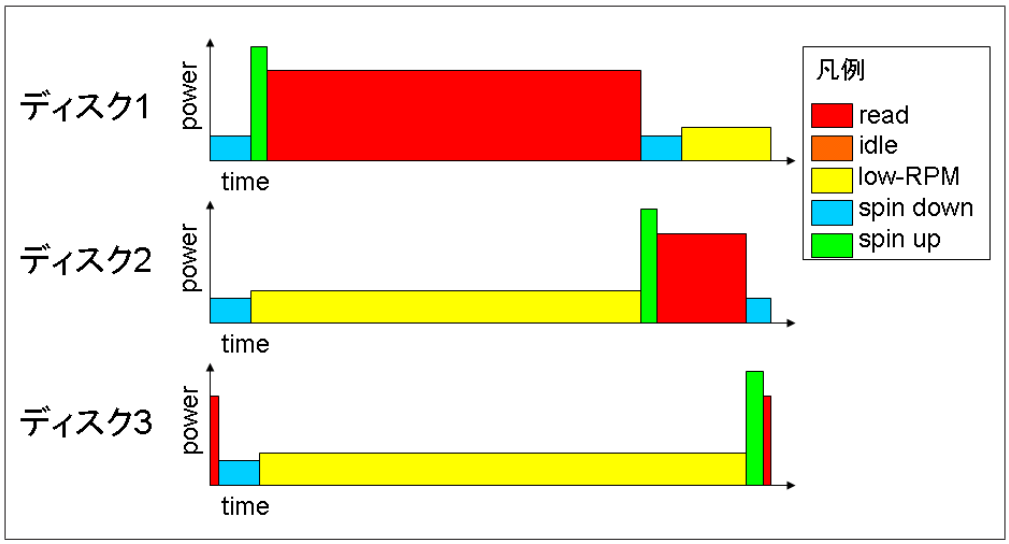

図 6.11: Q8 left deep PAC における各ディスクの動作状況

6.4 Q8 right deep

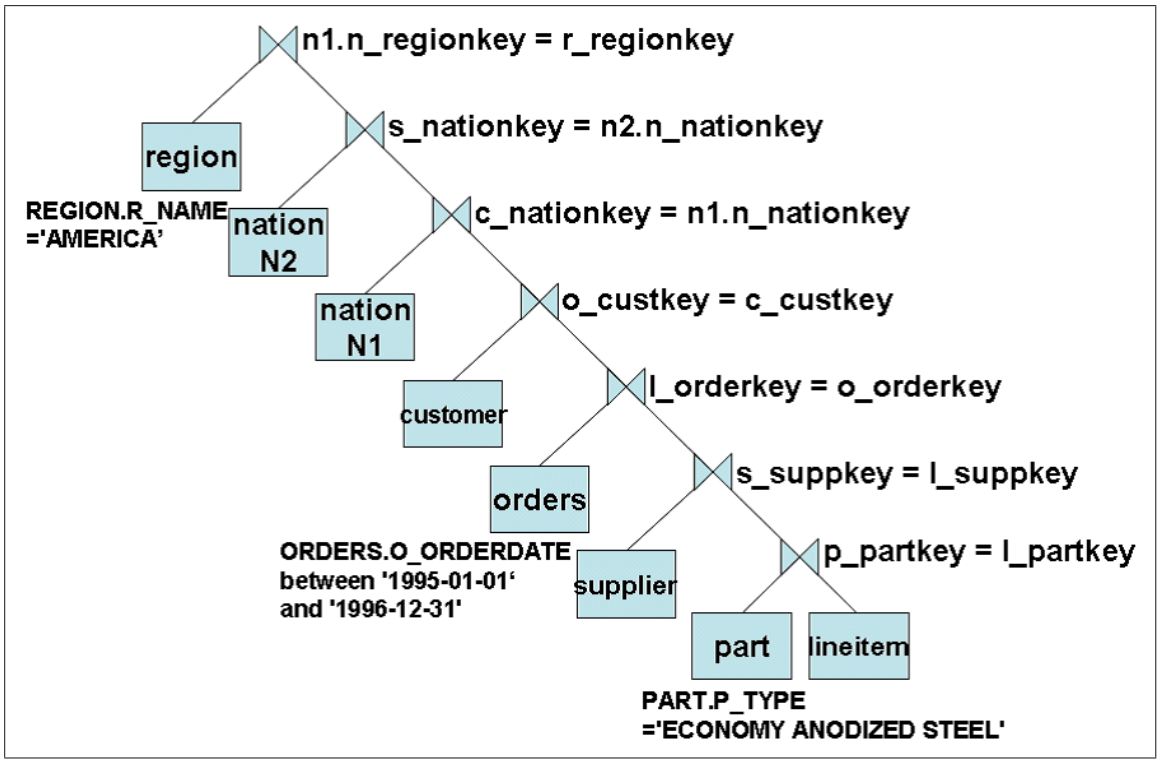

図 6.12: Q8 right deep の問い合わせ実行計画

図 6.12 は、Q8 right deep の問い合わせ実行計画の木構造を示したものである。これを元にして問い合わせを実行すると、図 6.13 のようなディスクアクセスを行う。

right deep はまず build を一度に行ってしまうが、テーブルの順序は任意である。今回の場合は、DBMS によって図 6.13 のような順番に最適化された。最後に、probe のために lineitem テーブルを読み込んでいる。

これから得られたスループットをモデルに適用し、電力を計算すると、表 6.2 のような結果が得られた。

表 6.2 を元に、NC、TPM、PAC におけるそれぞれのディスクの電力を、状態遷移コストも含めて計算すると、図 6.14、図 6.15 のようになる。それぞれ、省電力モードとして low-RPM standby、standby を用いている。

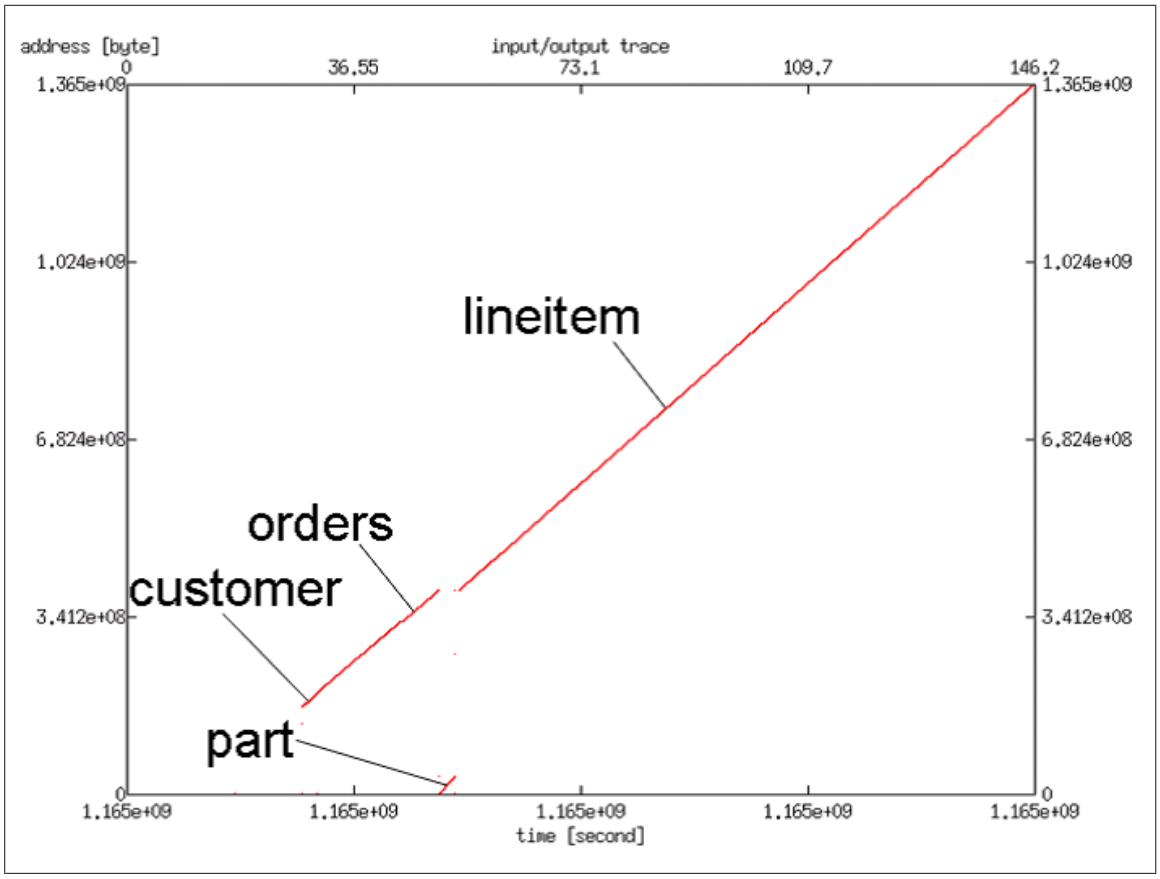

図 6.13: Q8 right deep のディスクアクセス

表 6.2: Q8 right deep の各フェーズにおける電力量

table	type	time [s]	size [MB]	throughput	power [W]	energy [J]
part	build	2.391	31.605	13.218	5.959	14.248
lineitem	probe	93.353	973.898	10.432	5.764	538.057
orders	build	19.628	196.411	10.007	5.734	112.545
customer	build	2.358	27.705	11.749	5.856	13.808
others	others	0.190				
total		117.920	1229.619			678.658

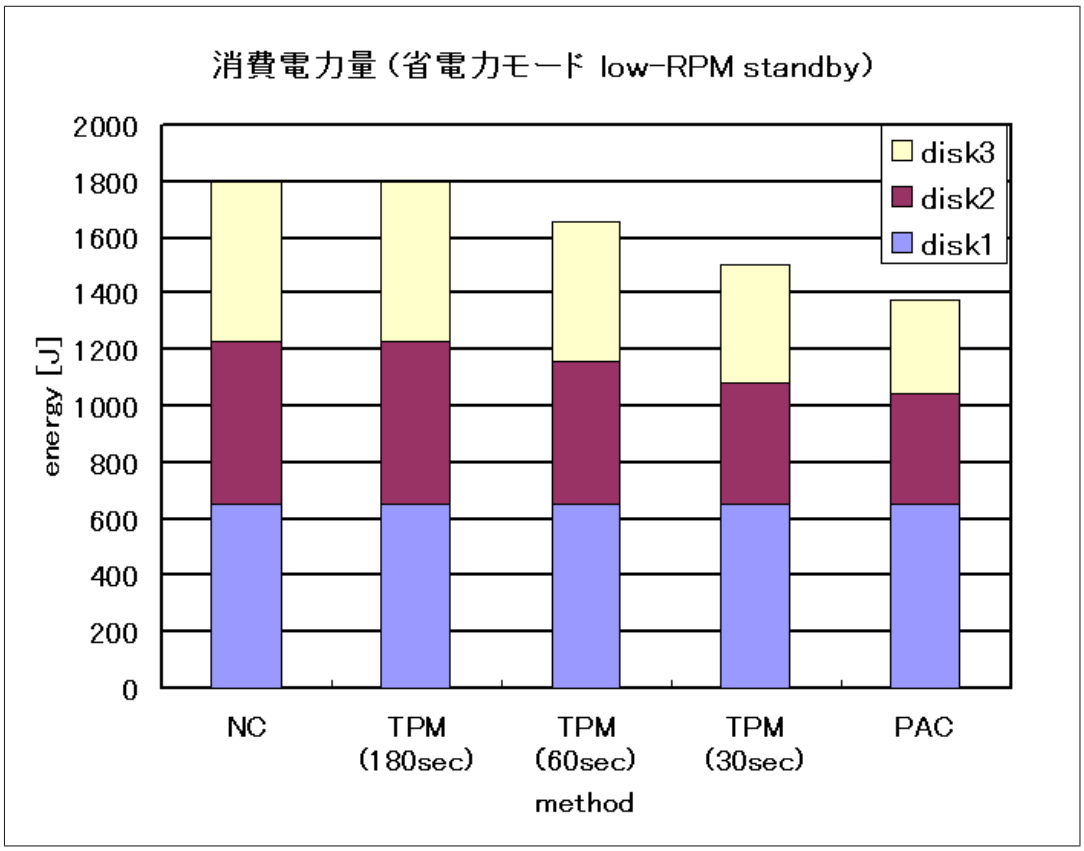

図 6.14: Q8 right deep における各制御方式の消費電力量 (省電力モード low-RPM standby)

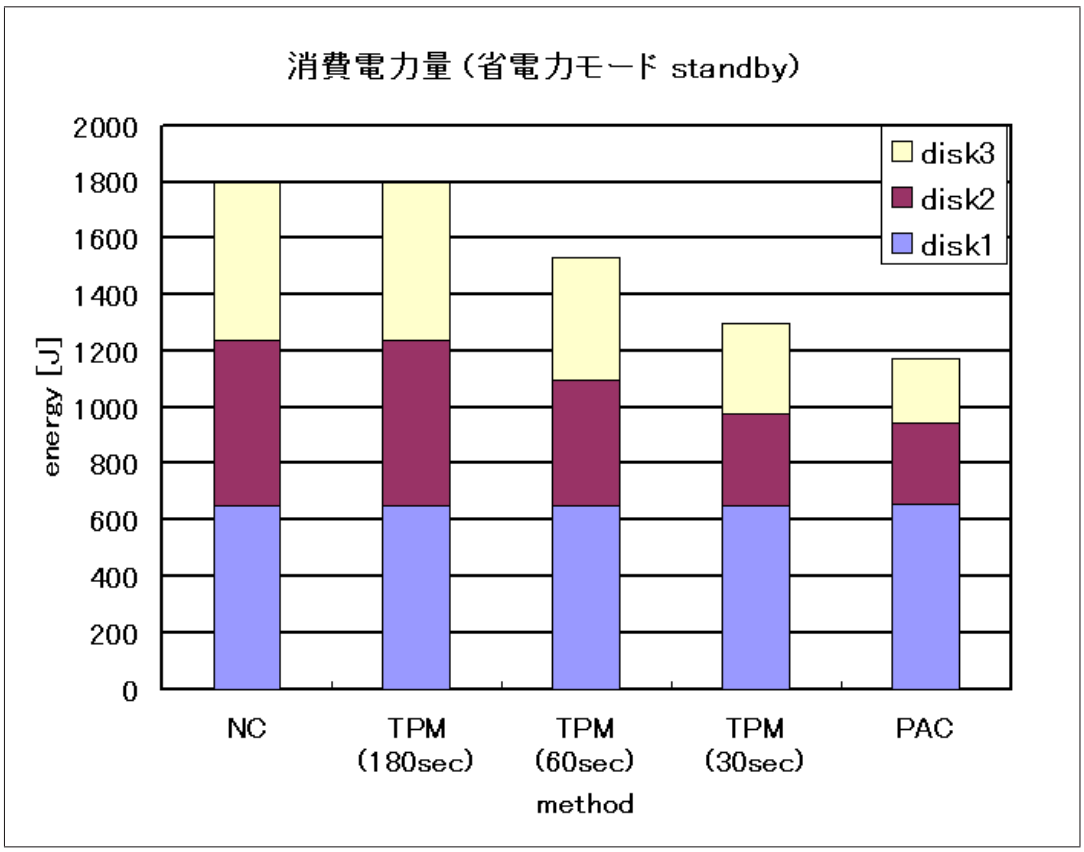

図 6.15: Q8 right deep における各制御方式の消費電力量 (省電力モード standby)

6.5 Q9 left deep

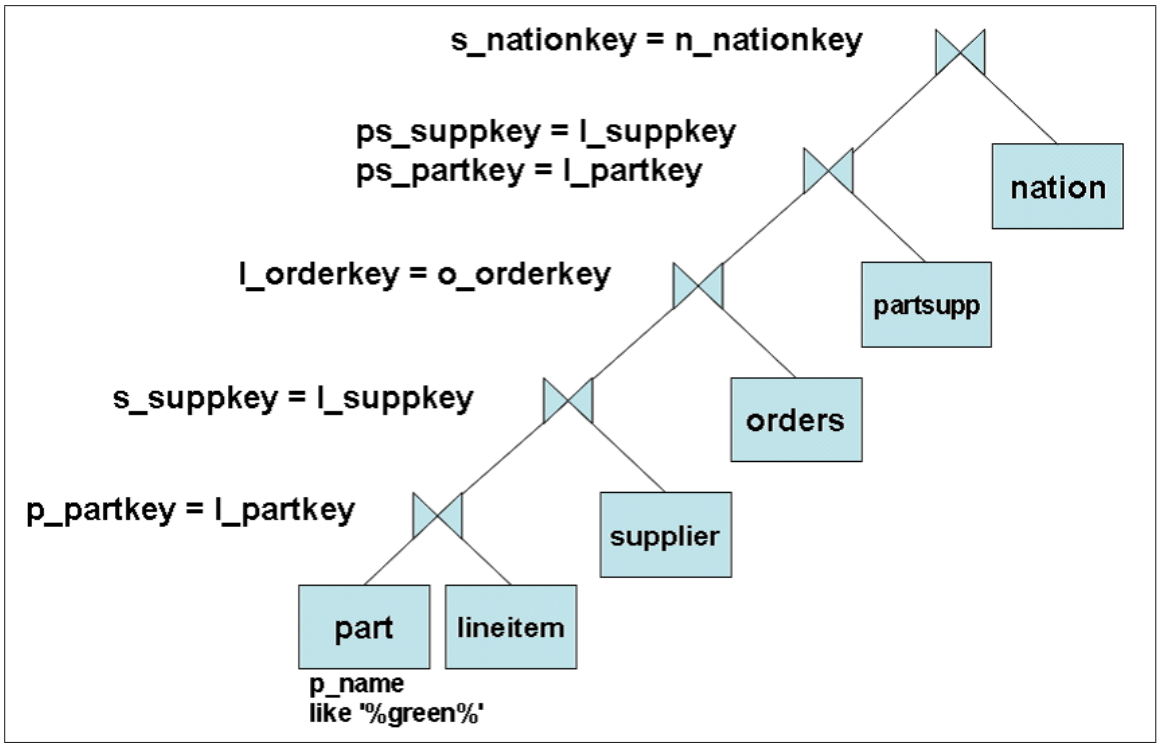

図 6.16: Q9 left deep の問い合わせ実行計画

図 6.16 は、Q9 left deep の問い合わせ実行計画の木構造を示したものである。これを元にして問い合わせを実行すると、図 6.17 のようなディスクアクセスを行う。

Q9 になると、各フェーズの間にディスクアクセスをしていない時間が多く見られるようになる。これは、probe の時間と、Join の結果で得られたテーブルを新たに build している時間と考えられ、そのテーブルの大きさは、Q8 の時よりも大きくなっている (Join の条件に合うタプルが多くなっている) と考えられる。この時間は、idle 状態として電力量に加算することにする。

これから得られたスループットをモデルに適用し、電力を計算すると、表 6.3 のような結果が得られた。

表 6.3 を元に、NC、TPM、PAC におけるそれぞれのディスクの電力を、状態遷移コストも含めて計算すると、図 6.18、図 6.19 のようになる。それぞれ、省電力モードとして low-RPM standby、standby を用いている。

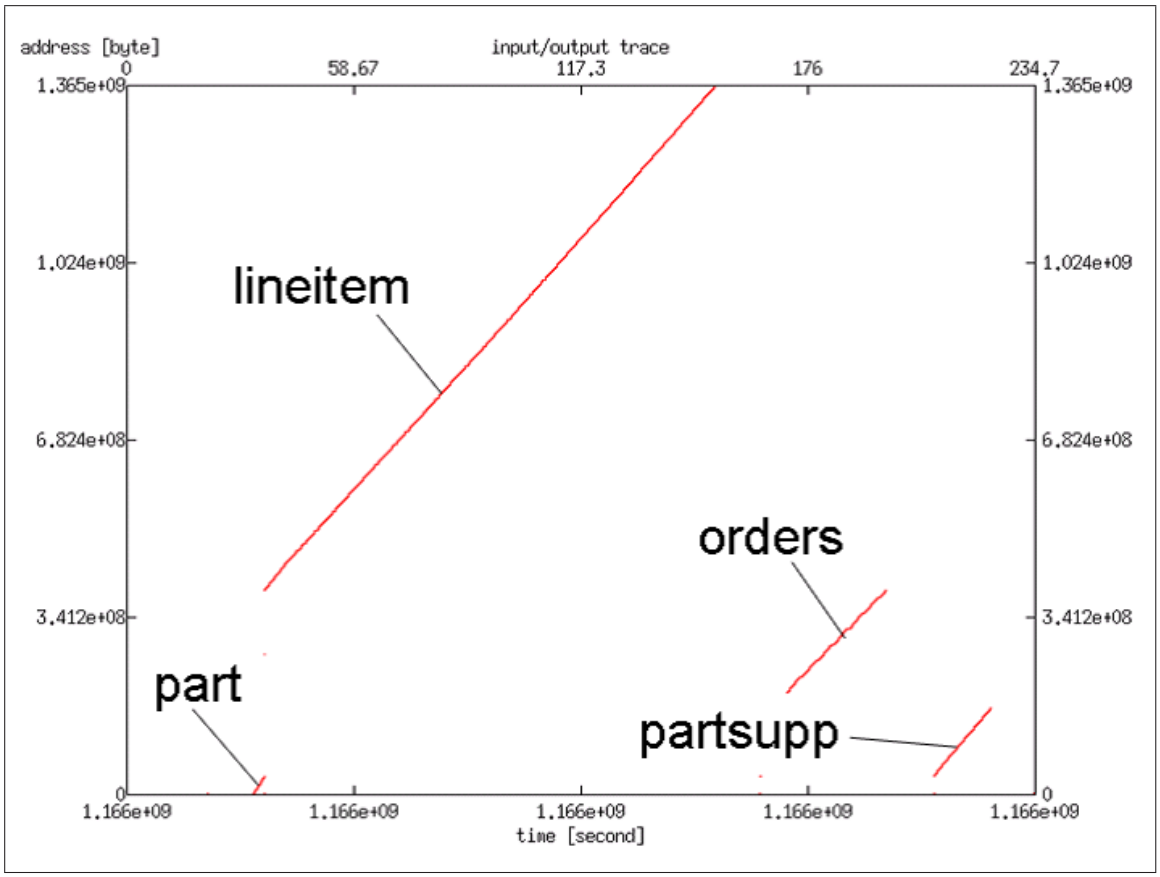

図 6.17: Q9 left deep のディスクアクセス

表 6.3: Q9 left deep の各フェーズにおける電力量

table	type	time [s]	size [MB]	throughput	power [W]	energy [J]
part	build	2.652	31.605	11.917	5.868	15.561
lineitem	probe	116.760	973.898	8.341	5.617	655.853
orders	probe	25.549	196.411	7.688	5.571	142.341
partsupp	probe	14.685	130.793	8.907	5.657	83.069
others	others	56.970(idle)			4.744(idle)	270.293(idle)
		0.274(other)				
total		216.890	1332.707			1167.117

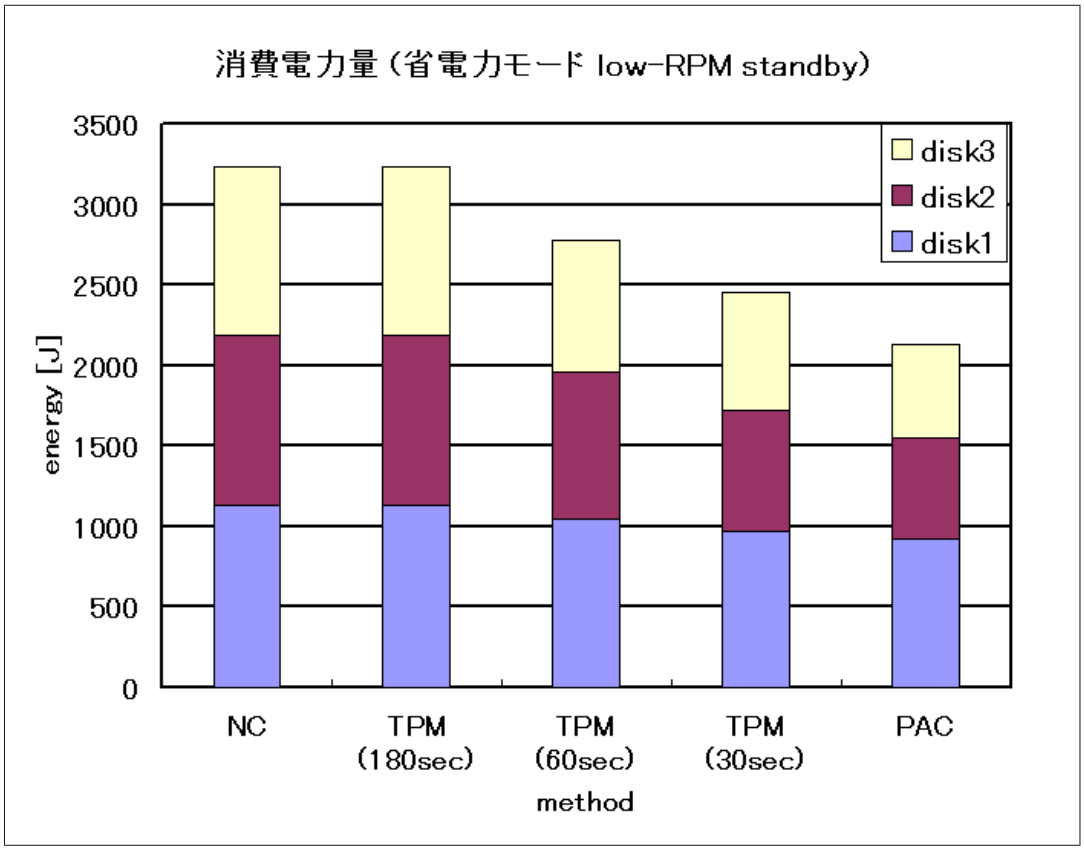

図 6.18: Q9 left deep における各制御方式の消費電力量 (省電力モード low-RPM standby)

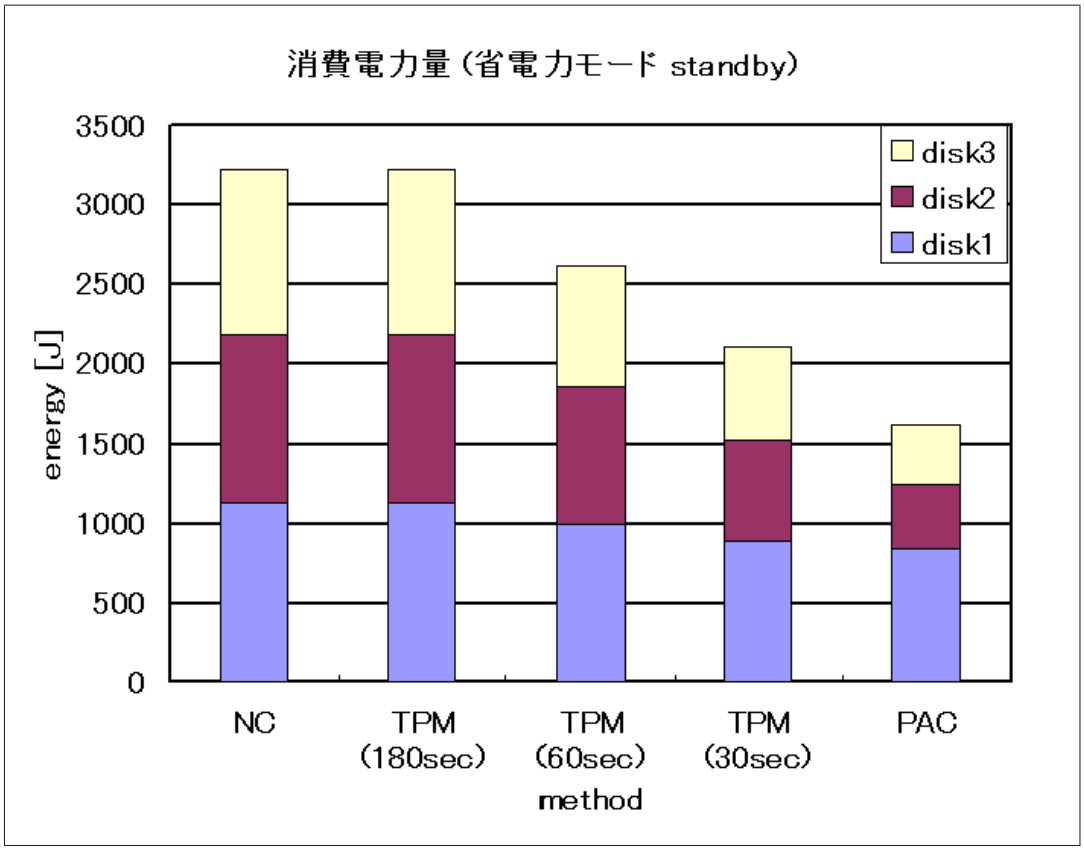

図 6.19: Q9 left deep における各制御方式の消費電力量 (省電力モード standby)

6.6 Q9 right deep

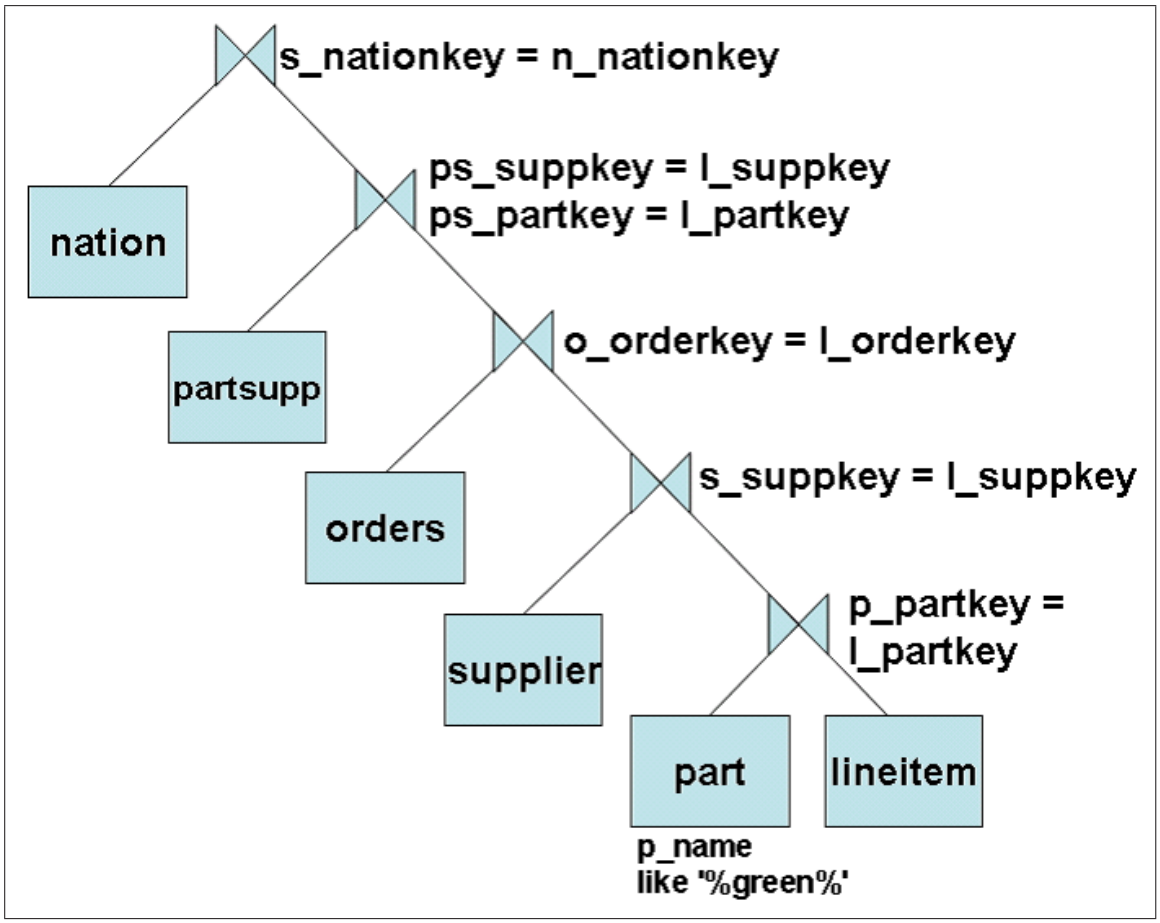

図 6.20: Q9 right deep の問い合わせ実行計画

図 6.20 は、Q9 right deep の問い合わせ実行計画の木構造を示したものである。これを元にして問い合わせを実行すると、図 6.21 のようなディスクアクセスを行う。形としては、Q8 right deep と類似しているが、lineitem テーブルによる probe の後に長い idle 時間が存在する。これは、Q9 left deep における、各フェーズ間の idle 状態が、right deep では最後にまとまったと考えられる。

これから得られたスループットをモデルに適用し、電力を計算すると、表 6.4 のような結果が得られた。

表 6.4 を元に、NC、TPM、PAC におけるそれぞれのディスクの電力を、状態遷移コストも含めて計算すると、図 6.22、図 6.23 のようになる。それぞれ、省電力モードとして low-RPM standby、standby を用いている。

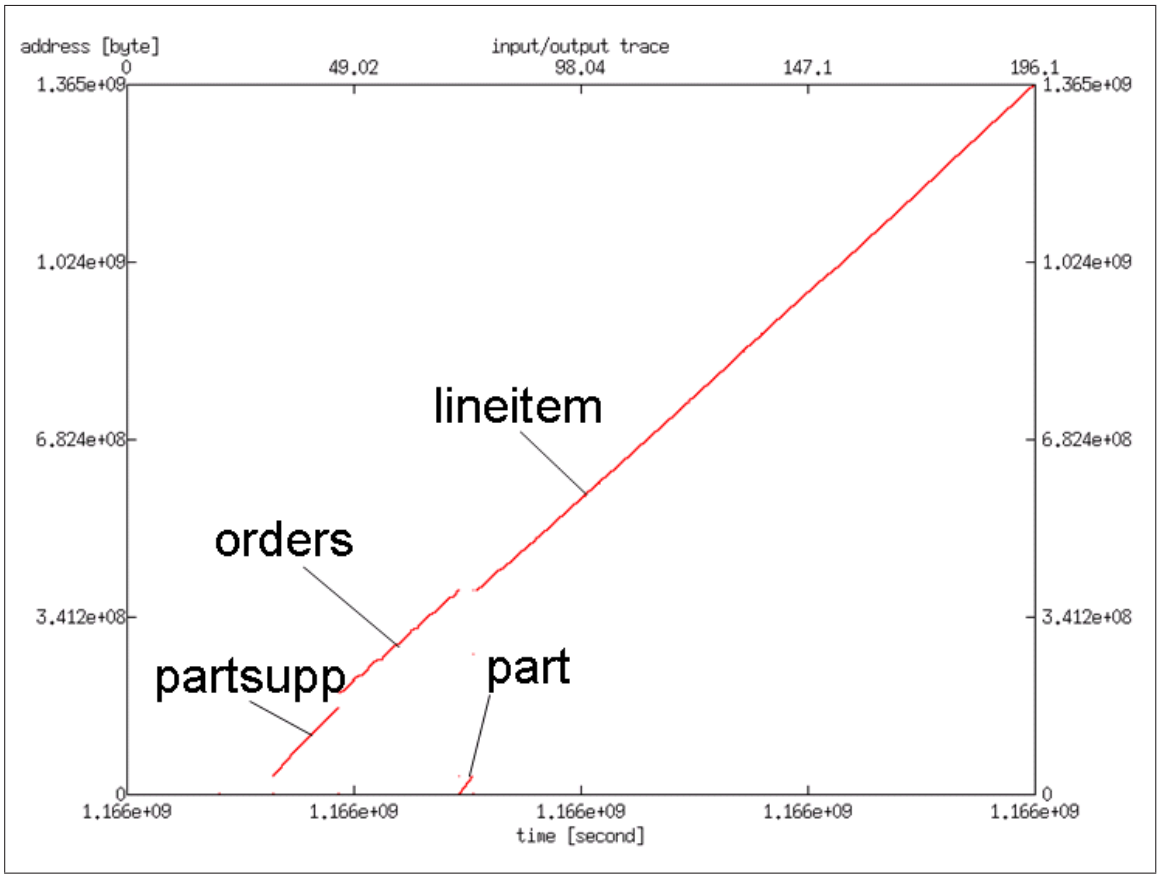

図 6.21: Q9 right deep のディスクアクセス

表 6.4: Q9 right deep の各フェーズにおける電力量

table	type	time [s]	size [MB]	throughput	power [W]	energy [J]
part	build	2.775	31.605	11.389	5.831	16.180
lineitem	probe	121.276	973.898	8.030	5.595	678.580
orders	build	25.927	196.411	7.576	5.563	144.243
partsupp	build	13.978	130.793	9.357	5.688	79.511
others	others	42.580(idle)			4.744(idle)	202.022(idle)
		0.325(other)				
total		206.861	1332.707			1120.536

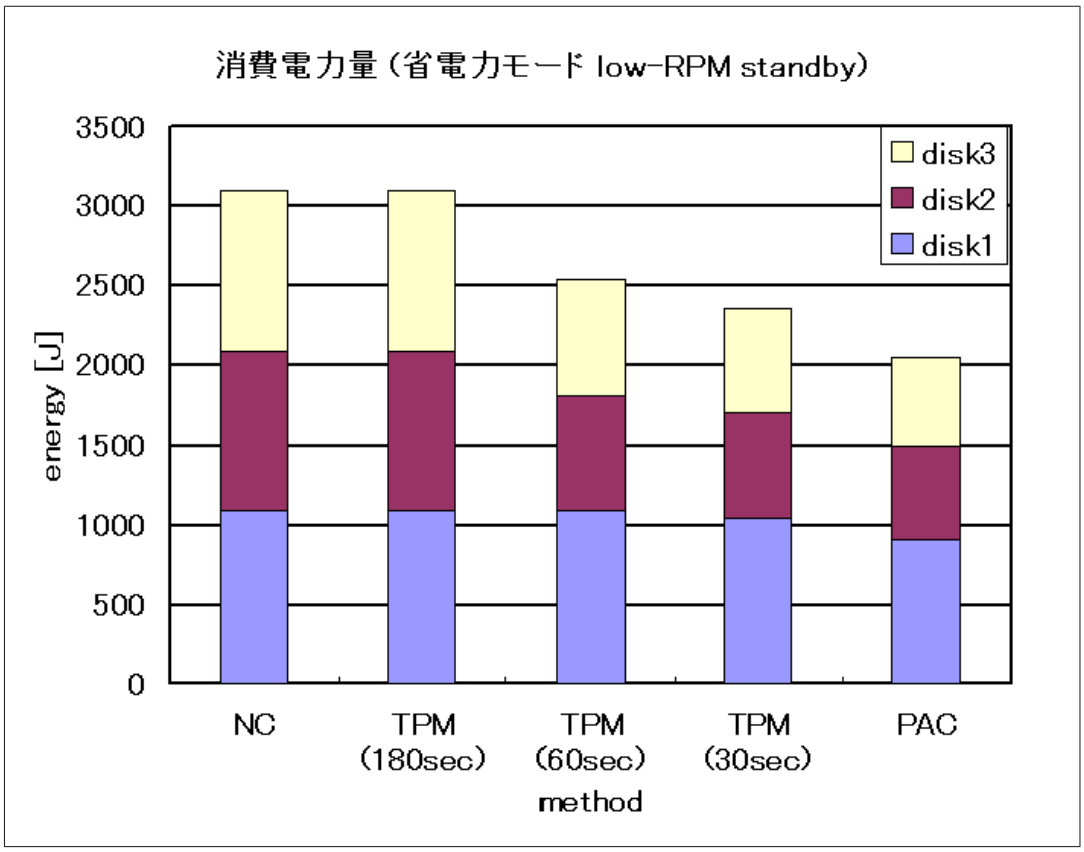

図 6.22: Q9 right deep における各制御方式の消費電力量 (省電力モード low-RPM standby)

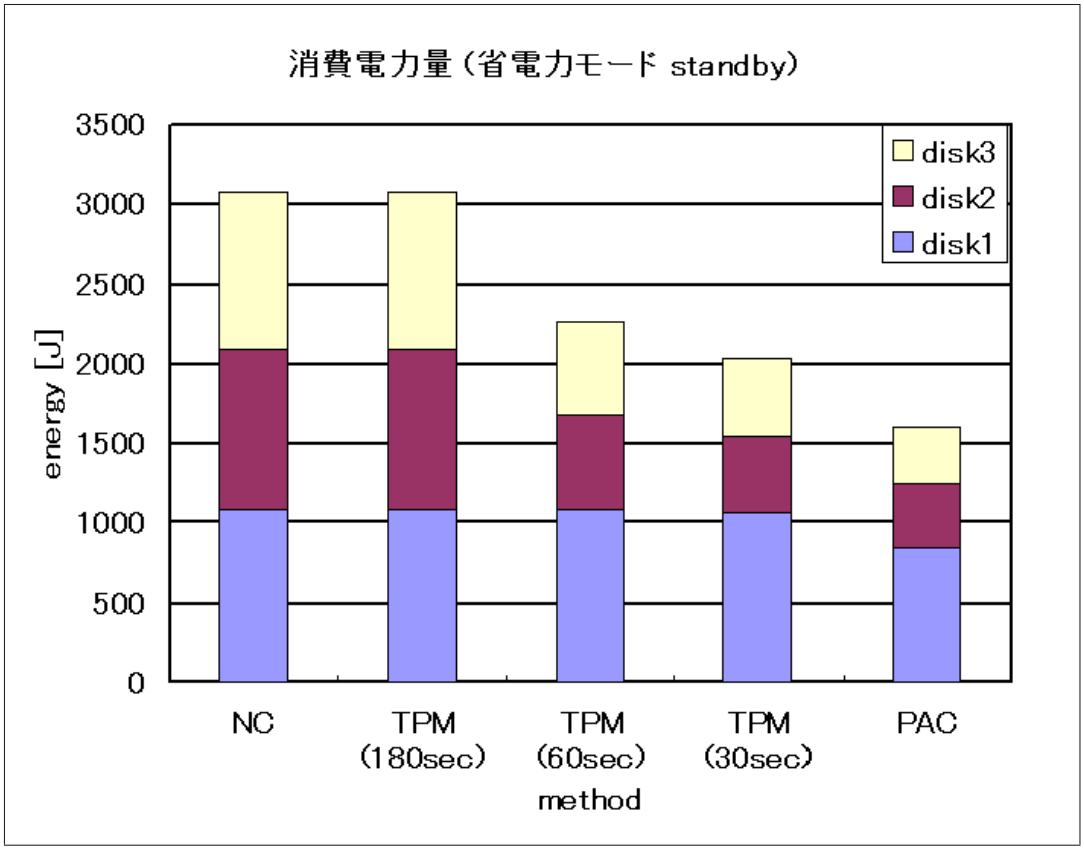

図 6.23: Q9 right deep における各制御方式の消費電力量 (省電力モード standby)

6.7 実験結果のまとめ

図 6.24、図 6.25 はそれぞれ、省電力モード low-RPM standby、standby を用いて各問い合わせにおける、TPM と PAC を NC と比較した時の電力量削減率をまとめたものである。また、各問い合わせの実行時間は、図 6.26、図 6.27 のようになっている。

Q8 の省電力モード standby では、TPM で最大 27 %、PAC で最大 35 %の削減電力量が得られ、Q9 の省電力モード standby では、TPM で最大 35 %、PAC で最大 50 %の削減電力量が得られた。Q9 では、Join の際の条件に当てはまるタプルが多かったと見られ、left deep での probe と再 build の時と、right deep での probe の際にディスクアクセスのない idle 状態が多く見られたため、その時の電力量削減の割合が Q8 より大きくなる結果となった。

次に、spin up/down のコストに関して考察すると、TPC-H のように一つの問い合わせが非常に大きい場合において各問い合わせでそれぞれのディスクに課す spin up/down の回数が数回という規模は、数万回の spin up/down に耐えうるディスクとしてはかなり現実的な数値であると考えられる。また、全体の処理時間を考慮に入れると、数回の spin up/down は比較的小さな割合の追加コストであるということができる。

また、今回は Join の順序を DBMS の最適化にほぼ合わせたが、spin up/down の回数を最小にするような最適化を考えることにより、更に無駄を抑えることができると考えられる。加えて、テーブルを各ディスクにどのように配置するかも、spin up/down の回数に影響を与えることになる。

更に、今回の解析では、PAC の機能を、idle 期間を発見したら直ちに spin down するという極めて単純なものにした[4]。しかし、問い合わせ実行計画の情報を利用することで、更に多機能な制御が可能となる。例えば、あるディスク A の読み込みが完了する前に次に読むべきディスク B を spin up し始め、ディスク A の読み込み完了とディスク B の spin up 完了が同時になるように制御すれば、実質的な spin up の時間的コストは 0 にすることができる。また、ある idle 期間が閾値時間以下と予想されるならば spin down を行わず、閾値時間以上と予想されるならば idle 期間に入ると同時に spin down を行うようにすれば、idle 期間に入ると同時に spin down することができない TPM との性能差を更に広げることが可能になると推測される。

このような PAC の多機能化に関しては今後の課題となる。

[4] これは、TPM の閾値 0sec と実質的に同じである。

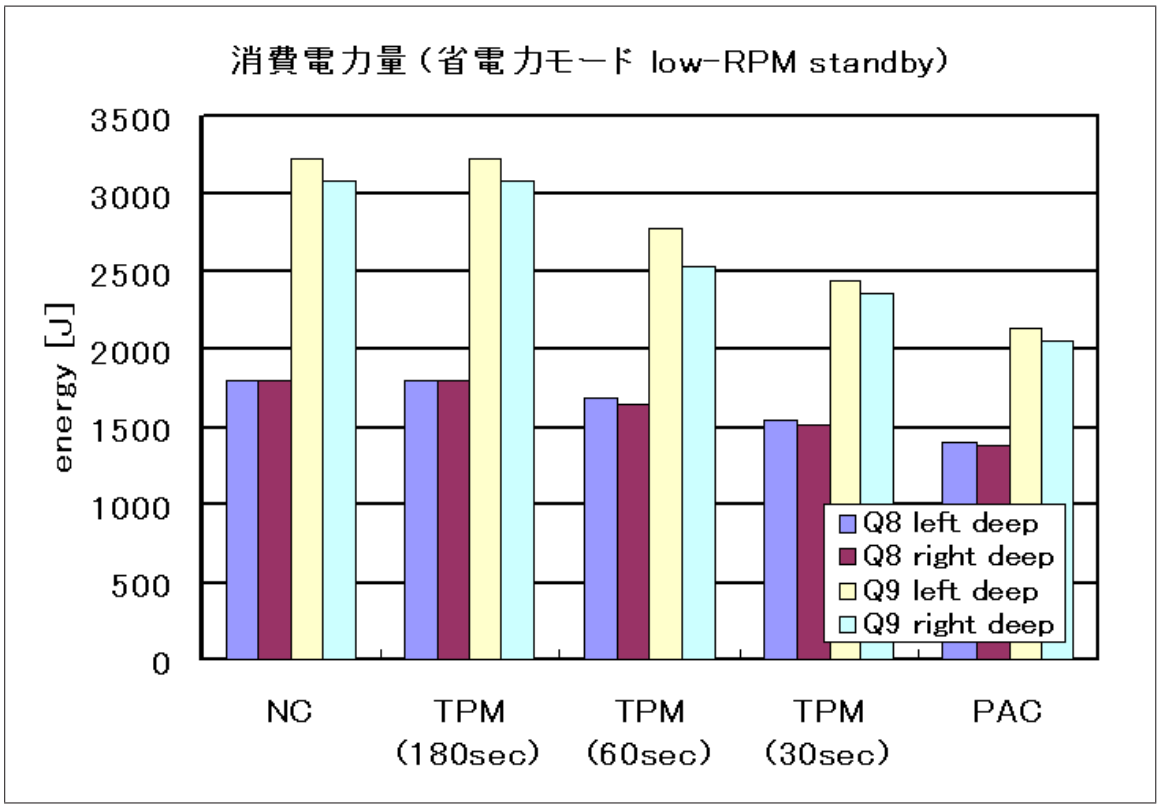

図 6.24: 各方式の電力量の比較 (省電力モード low-RPM standby)

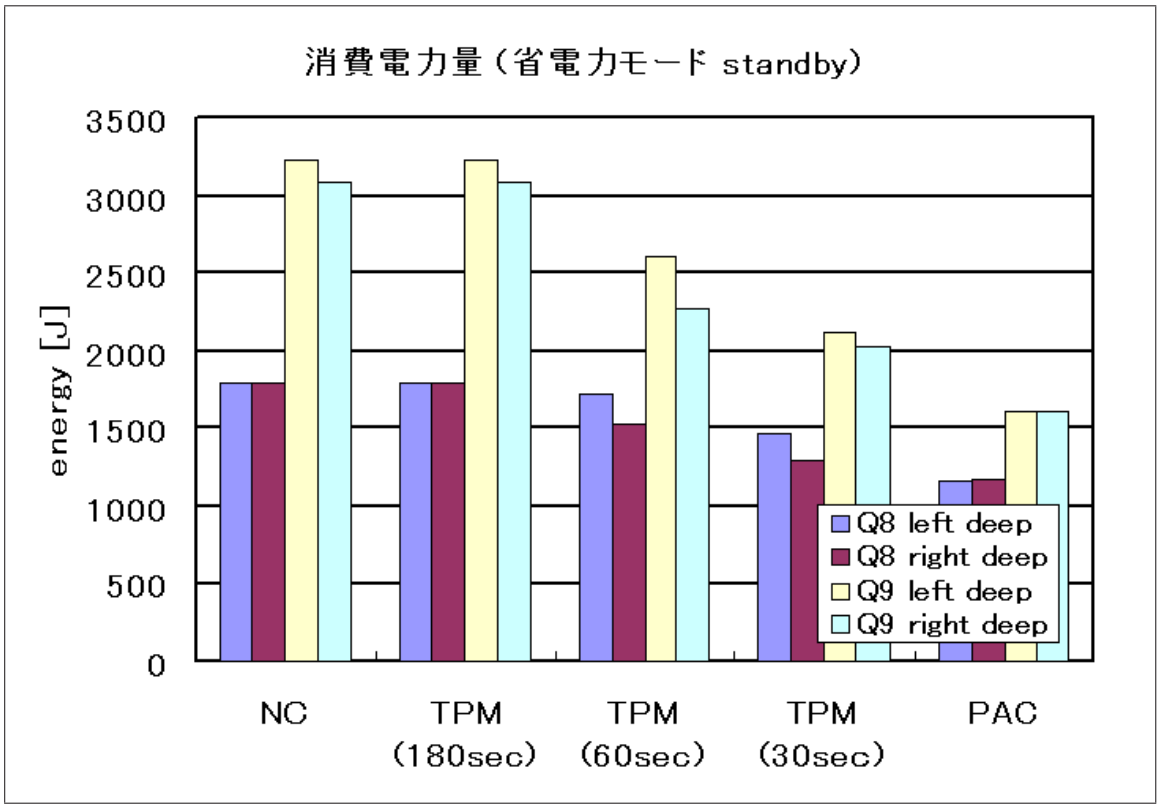

図 6.25: 各方式の電力量の比較 (省電力モード standby)

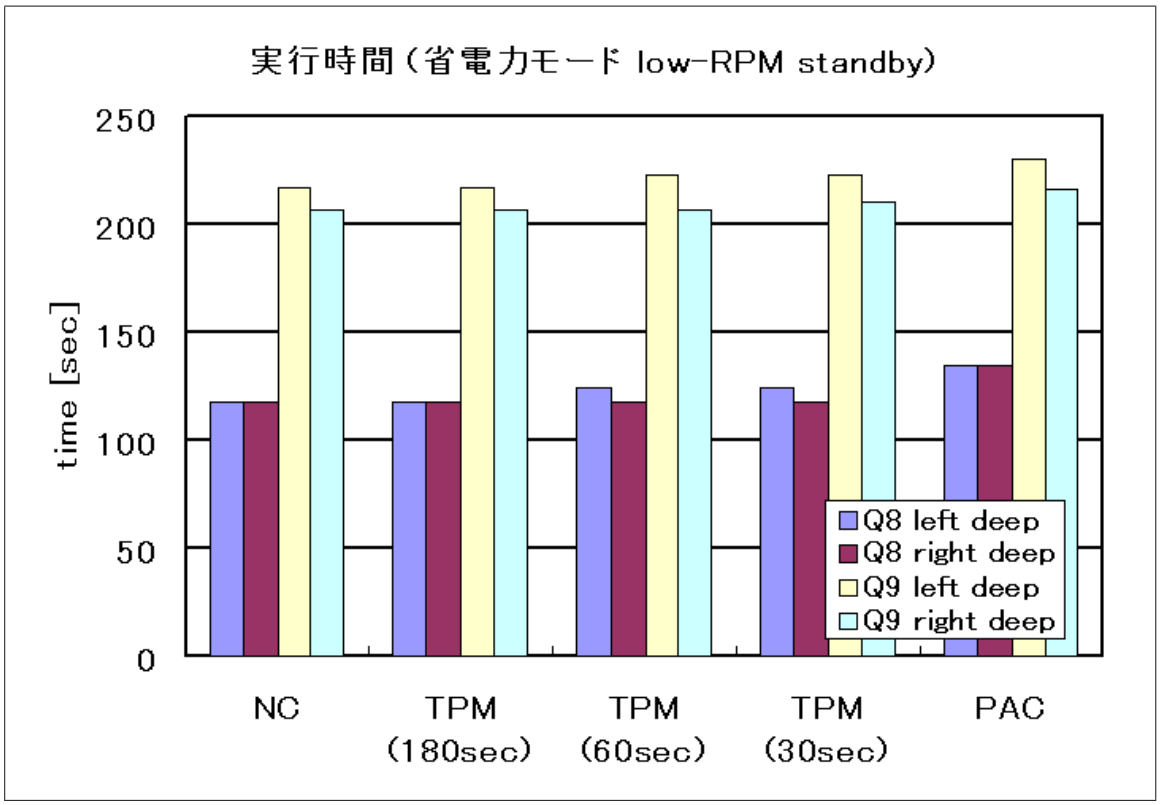

図 6.26: 各問い合わせの実行時間 (省電力モード low-RPM standby)

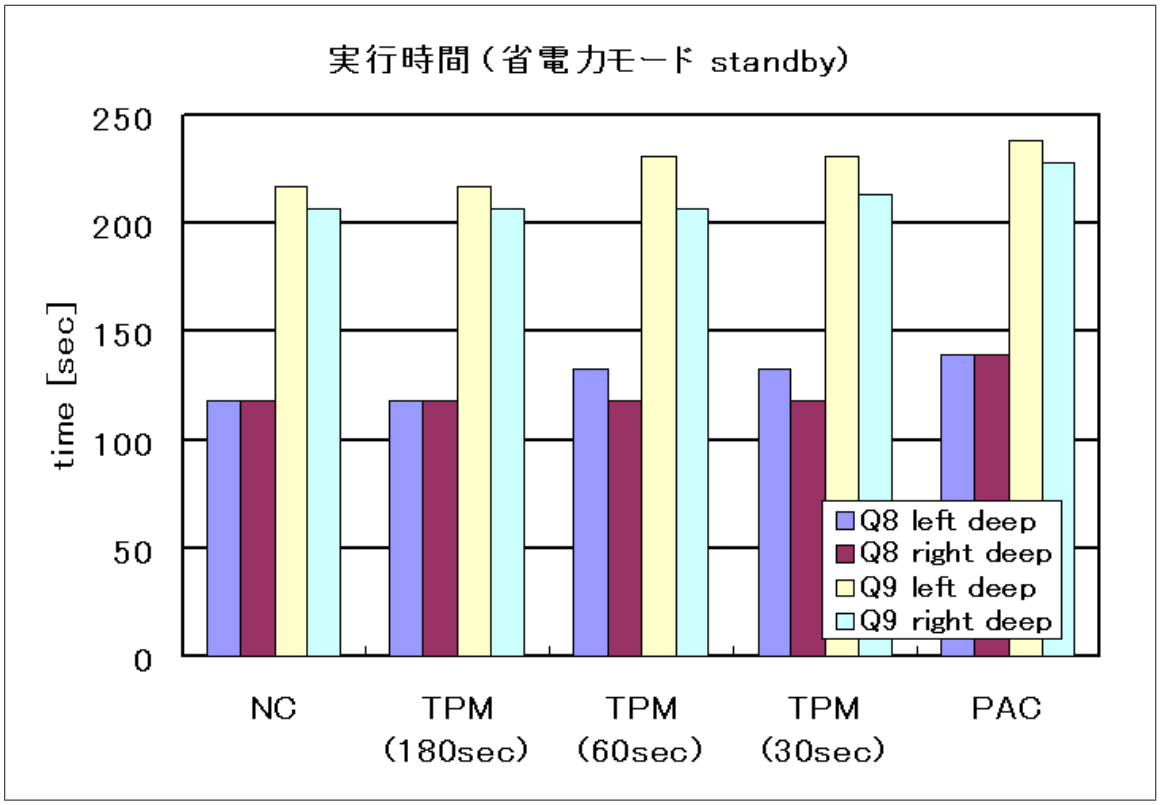

図 6.27: 各問い合わせの実行時間 (省電力モード standby)

第7章　まとめと今後の課題

7.1　まとめ

本研究では、巨大なデータベースシステムにおけるディスクアレイの消費電力問題を背景とし、問い合わせ実行計画に重点を置いたディスクアレイ全体を対象とした省電力の方式を提案した。特に複雑な問い合わせ処理を行う DSS に関しては、データベーステーブルの Join に Hash Join を用いることで、長時間の idle 状態に置かれるディスクが多数存在することに着目し、それらのディスクを省電力モードにすることで電力削減を図った。また、データベーステーブルの特徴にあわせて問い合わせ実行計画を工夫しながら、left deep Hash Join、right deep Hash Join に関して解析を行った。

まず、独自の I/O 負荷生成ツールを用いてディスクを動作させ、実際にディスクの消費電力を測定することにより、スループットに対するディスクの消費電力の関係をモデル化した。これは、過去の研究ではほとんど見られないものである。結果は、スループットの増加に対してほぼ直線的な電力上昇が見られ、I/O のサイズが大きいほど電力効率が良いことがわかった。特に、シーケンシャルモデルに関しては正確なモデルを得ることができた。

次に、問い合わせ実行計画を用いることで読み込むテーブルの順序が確定すると、アプリケーション、DBMS、OS から能動的にディスクの動作を決定することで、より効率的な省電力が実現できるということを示し、その環境を想定した三つの省電力化方式を提案した。そして、先の消費電力モデルに加え、normal idle 状態、low-RPM standby 状態、standby 状態の電力測定値を用い、三つのディスクからなるデータベースを想定した省電力化方式の消費電力量を算出した。具体的には、独自の I/O 監視ツールを用いて、各テーブルを読み込む際のスループットを得た後、モデルに適用して予想消費電力を得る。更に、read、idle 状態の時間とそれぞれの予想消費電力の積を取ることにより、予想消費電力量を算出した。NC、TPM、PAC という三種類の方式を比較したところ、省電力モードとして standby を用いると、何も省電力の動作を行っていない NC に対して、TPM(閾値 30 秒) は 18〜34 %、PAC は 35〜50 %の電力量削減が得られることが分かった。

7.2　今後の課題

まず、実環境として全てのテーブルが一つのディスクに入っているものを構築した。よって、実際に複数のディスクにテーブルが分散されている環境で試してみる必要がある。また、今回

はベンチマークのスケールファクタを小さくし、小さいテーブルや細かい読み込みを無視した計算を行っていたため、それらも考慮に入れた正確な実験が今後必要となる。

テーブルのディスク上の配置や、spin up/down の回数を抑える最適化を考えること、PAC を更に高機能にすることによっても、更なる電力量の削減が得られるものと考えられる。また、シングルスレッドで一度に一つのディスクしかアクセスしていなかった点も、まだまだ展開の余地がある。

最後に、6.1 節で説明したように、モデルと問い合わせ実測値が若干異なっていたので、これらの原因を明らかにし、正確なモデル化を行うことや、ランダムアクセスに対するモデル化も課題となる。

謝辞

本研究を進める上でお世話になりました多くの方々に、この場を借りてお礼申し上げます。

まず、本研究室の指導教員である喜連川優教授には、研究テーマや研究への取り組みに関して多くの助言を頂き、大変感謝しております。

また、秘書の中野恵理さん、小笠原薫さん、井崎葉子さん、松島恵里さんは、事務の面で研究室を支えてくださり、快適な研究を行うことができました。ありがとうございます。

星野喬氏、合田和生氏は、データベース、ストレージ分野での研究を活かし、経験の少ない私に知識やアドバイスを与えてくださいました。本当に感謝しております。

山口実靖氏からは、研究室の計算機を整備するに当たり、計算機に関する技術を多く学びました。大変お世話になりました。

更に、研究室の先輩方、研究員の方々、同期の修論生の皆様にも、日常的な相談から技術的な指導まで、幅広く私の研究生活を支えて頂きました。

本論文を書き終えることができたのも、ひとえに皆様のおかげです。どうもありがとうございました。

参考文献

[1] Qingbo Zhu, Zhifeng Chen, Lin Tan, Yuanyuan Zhou, Kimberly Keeton, John Wilkes, "Hibernator: Helping Disk Arrays Sleep through the Winter", SOSP '05, October 23-26, 2005.

[2] White paper by Dennis Colarelli, Dirk Grunwald, Michael Neufeld, "The Case for Massive Arrays of Idle Disks (MAID)", Dept. of Computer Science, Univ. of Colorado, Boulder.

[3] White paper by Fred Moore, Aloke Guha, "Introducing COPAN Systems' MAID Architecture (Massive Arrays of Idle Disks)", HORIZON Information Strategies.

[4] White paper, "The Rise of MAID: A New Tier in Disk Storage", TANEJA GROUP.

[5] D. Colarelli and D. Grunwald, "Massive arrays of idle disks for storage archives", In Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, pages 1-11, June 2002.

[6] S. W. Son, G. Chen, M. Kandemir, "Disk Layout Optimization for Reducing Energy Consumption", ICS' 05, June 20-22.

[7] E. Carrera, E. Pinheiro, and R. Bianchini. "Conserving disk energy in network servers." In Proc. of the 17th International Conference on Supercomputing, June 2003.

[8] J. Wilkes, R. A. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID hierarchical storage system", ACM Transactions on Computer Systems, 14(1):108-136, 1996.

[9] Qingbo Zhu and Yuanyuan Zhou, "Power Aware Storage Cache Management", Department of Computer Science University of Illinois at Urbana-Champaign.

[10] H. Yada, H. Ishioka, T. Yamakoshi, Y. Onuki, Y. Shimano, M. Uchida, H. Kanno, and N. Hayashi, "Head Positioning Servo and Data Channel for HDD 's with Multiple Spindle Speeds", IEEE TRANSACTIONS ON MAGNETICS, VOL. 36, NO. 5, SEPTEMBER 2000.

[11] MySQL. http://www.mysql.gr.jp/

[12] PostgreSQL. http://www.postgresql.org

[13] TPC Benchmarks. http://www.tpc.org/

[14] TPC-H. http://www.tpc.org/tpch/spec/tpch2.5.0.pdf

[15] HGST Feature Tool.
http://www.hitachigst.com/hdd/support/downloads/FTool_User_Guide_203.pdf

[16] HGST Deskstar T7K250.
http://www.hitachigst.com/portal/site/jp/menuitem.e9e85a2f0b51ab518797c532aac4f0a0/

[17] 合田 和生, 喜連川 優, “ログ転送を用いたディザスタリカバリシステムにおけるディスクストレージの省電力化方式の検討”, DEWS2007.