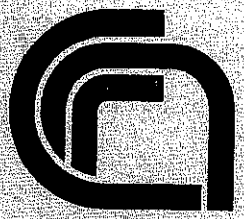
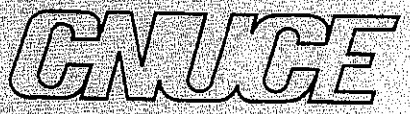


Consiglio Nazionale delle Ricerche

Il Clustering nelle Basi di Dati a Oggetti

Giovanni Mainetto, Giuseppe Rossini

Rapporto CNUCE-B4-1998-013



Pisa

Il Clustering nelle Basi di Dati a Oggetti

Giovanni Mainetto, Giuseppe Rossini
CNUCE CNR, Via Santa Maria 36, 56100 Pisa 56126 Italy
e-mail: G.Mainetto@cnuce.cnr.it, Rossini@gmsun.cnuce.cnr.it

Abstract: *Questo articolo di rassegna illustra il problema del clustering relativamente ai sistemi per la gestione di basi di dati orientate a oggetti. Il problema del clustering è quello di decidere come porre gli oggetti nelle pagine fisiche della base di dati in modo tale che, dato un insieme di applicazioni, il numero dei fault di pagina risulti minimo. In un sistema per la gestione di basi di dati orientate a oggetti, fornire una "buona" soluzione a questo problema è piuttosto complesso in virtù della grande espressività del modello dei dati – assai distante quindi dalla rappresentazione fisica del sistema a oggetti -, delle diverse modalità di accesso agli oggetti – cioè navigazionale e associativo – da parte delle applicazioni e della complessità dell'architettura dei sistemi a oggetti – tipicamente cliente-servente. Due classi di soluzioni sono emerse in letteratura: quelle basate su un approccio ingegneristico e quelle basate su un approccio matematico. Le varie soluzioni tecniche che si trovano in letteratura e i parametri in gioco sono presentati in dettaglio.*

Abstract: *This review paper deals with the clustering problem in object oriented database management systems. The clustering problem can be defined in this way: given a set of database applications, find a placement of objects on physical pages such that the number of page faults is minimal. In an object-oriented database management system, the task of finding a fine solution of the problem is rather complex because of the greath expressiveness of data model – which is distant from the physical representation of persistent objects -, different ways in which database applications access objects – i.e., navigational and associative -, and complexity of object-oriented database's architecture – typically client-server. Two classes of solutions approach appeared in literature: the engineering approach and the mathematical approach. This paper present details the clustering problem, the most representative technical solution emerged in literature, the parameters involved.*

Categories and Subject Descriptors: H, H2, H2.2, H3.2, H3.3

Keywords: Clustering, Object-oriented databases; Physical database design; Performance evaluation

1. Introduzione

Con il termine *clustering*[1] si intende quel modo che un *Sistema per la Gestione di Basi di Dati* (*SGBD*) ha di memorizzare i dati in memoria permanente volto a far sì che dati persistenti differenti acceduti dalle applicazioni in tempi estremamente ravvicinati siano posizionati in spazi di memoria permanente contigui. La finalità di questa memorizzazione è quella ridurre i tempi di accesso ai dati persistenti da parte delle applicazioni, anticipandone in qualche modo la presenza in memoria temporanea attraverso un precaricamento. In termini di tecnologia dei dischi, il clustering permette di ridurre i tempi di

[1] Alcuni autori tendono a tradurre in italiano *clustering* con raggruppamento. Noi lasceremo il termine in inglese.

lettura agendo soprattutto sul *tempo di posizionamento* delle testine. In termini di *gestione dei buffer* da parte del SGBD, il clustering mira a ridurre i *fault di pagina dei dischi*. Esiste una evidente analogia fra il *principio di località* dei dati sfruttato nella memoria virtuale dei sistemi operativi e gli effetti che si vogliono ottenere tramite il clustering sulle applicazioni che utilizzano basi di dati. Questa analogia diventa una completa identificazione in quei Linguaggi per la Programmazione Persistente che fanno uso di una memoria virtuale permanente come unico supporto di memorizzazione dei dati persistenti.

Con l'avvento dei SGBD *Orientati a Oggetti* (*SGBDOO*) e delle architetture distribuite su rete locale, in particolare quelle cliente-servente, il problema del clustering è cresciuto in complessità sia per l'accresciuto numero di componenti architetturali in gioco, quali ad esempio quelli che si occupano della comunicazione fra i sottosistemi che si interfacciano sulla rete locale, sia per la maggior complessità dei singole componenti architetturali, quali ad esempio gli strumenti di analisi statica delle parti procedurali di una applicazione.

Esistono due approcci al problema del clustering nei SGBDOO, che hanno ovviamente aree di sovrapposizione. Il primo approccio, che definiremo *ingegneristico*, si pone il problema dal punto di vista del progettista della base di dati che deve dunque: capire quali sono i parametri che entrano in gioco nel problema in modo da scegliere la strategia più adatta alle esigenze per cui è stato concepito il prodotto; essere in grado di ottenere prestazioni migliori nell'esecuzione delle applicazioni; avere la capacità di quantificare i miglioramenti delle prestazioni introdotti attraverso misure oggettive. Il secondo approccio è quello *matematico*. La base di dati a oggetti viene qui vista come un grafo orientato i cui nodi sono gli oggetti ed i cui archi sono le associazioni fra oggetti, rappresentati dagli *identificatori di oggetti* (*OID*). Il problema del clustering in questo caso viene ricondotto ad un problema di ottimizzazione su un grafo, la cui soluzione consiste nel trovare un suo partizionamento in sottografi, ognuno da assegnare a una pagina fisica della base di dati, in modo tale che per un dato insieme di applicazioni il numero di fault di pagina sia minimo. Dal momento che il problema è computazionalmente molto complesso, rientra infatti nella categoria dei problemi *NP-ardui*, per la sua soluzione vengono utilizzate tecniche che approssimano il partizionamento ottimo sulla base di diversi fattori di costo.

Questo articolo di rassegna illustra il problema del clustering, le varie soluzioni tecniche che si trovano in letteratura e i parametri in gioco. L'articolo è organizzato come segue: nel prossimo capitolo si introduce il problema del clustering, illustrato da due classici esempi, semplici ma significativi. Successivamente analizzeremo le due classi di soluzioni che si delineano in letteratura: quelle basate su un approccio ingegneristico e quelle basate su un approccio matematico. Infine proporremo le nostre considerazioni conclusive.

2. Il problema del clustering

Come precedentemente accennato, il problema del clustering consiste nel trovare un posizionamento degli oggetti all'interno delle pagine della base di dati in modo tale che sia minimizzato il numero totale di *fault di pagina* durante l'esecuzione delle applicazioni. Questo problema, che risulta in realtà assai complesso, può essere illustrato da alcuni semplici ma significativi esempi.

Consideriamo dapprima le due classi qui sotto descritte.

```
class Vertice {
        // coordinate di un punto nello spazio
        x, y, z: float;
};
```

```
class Cilindro {
        centro1: Vertice;
        centro2: Vertice;
        raggio: float;
};
```

Un oggetto di tipo *Cilindro* è costituito da due oggetti, centro1 e centro2, di tipo *Vertice* e da un raggio di tipo *float*; un oggetto di tipo *Vertice* viene rappresentato attraverso le coordinate *x,y,z* di un punto nello spazio tridimensionale.

La descrizione completa di un *Cilindro* consta dunque di tre oggetti. Consideriamo ora un'istanza di questo schema in cui l'oggetto *Cilindro*, di identificatore id_1, ha come componenti gli oggetti *Vertice* di identificatori id_2, id_3. Se id_1, id_2 e id_3 vengono memorizzati in differenti pagine del disco, come mostrato in Figura 1 (a), sono necessari tre accessi alla memoria secondaria per la ricostruzione della rappresentazione del cilindro id_1 in memoria principale. Assumendo come tempo medio di accesso al disco 10 ms per pagina, avremo che l'intero procedimento richiede 30 ms. Il risultato dell'operazione di lettura è mostrato nella parte (b) della Figura 1.

Poiché gli oggetti coinvolti non sono molto grandi, è possibile memorizzarli all'interno di una singola pagina. In questo caso è necessario un solo accesso

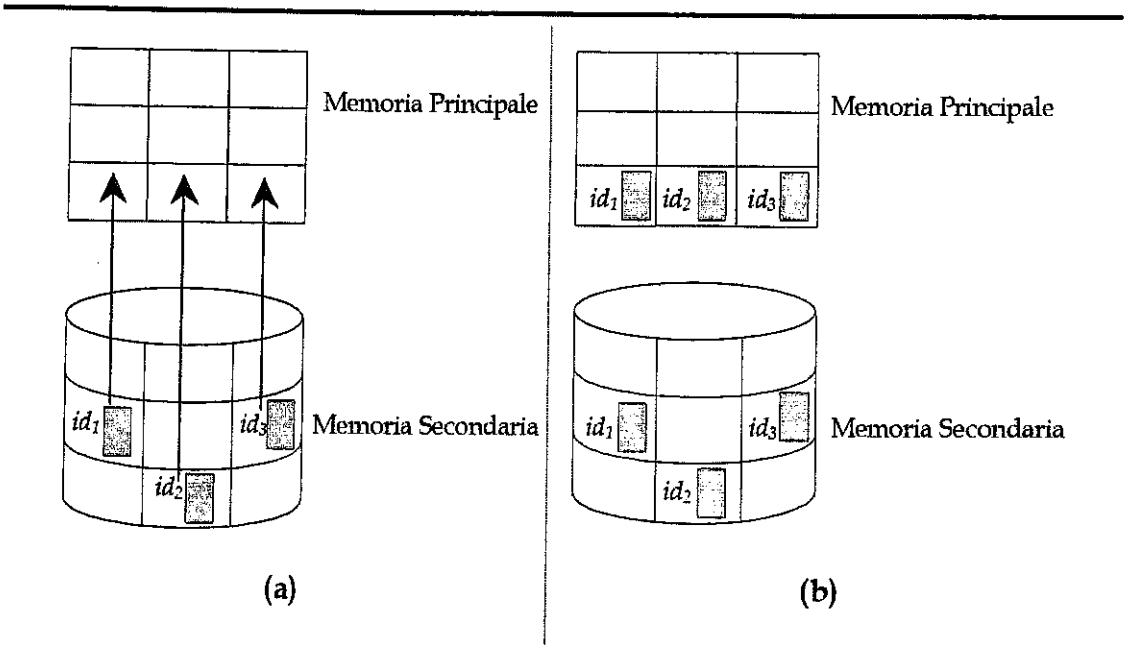


Figura 1: Distribuzione di un oggetto di tipo *Cilindro* in pagine disco

per portare in memoria principale l'intera rappresentazione di id_1 con costo 10 ms (1/3 del tempo necessario in precedenza). Ovviamente, avendo più oggetti correlati nella stessa pagina, diminuisce il numero delle letture da disco con conseguenti vantaggi per le prestazioni.

Ci sono altri vantaggi meno evidenti derivanti dal raggruppamento di oggetti correlati in unità fisiche. Quando si legge una pagina del disco, essa viene memorizzata in un *buffer*, cioè in una zona della memoria principale. Se sono necessarie molte pagine, essendo questa memoria limitata, si dovrà procedere ad una continua riscrittura e rilettura di queste dal disco, nonostante gli oggetti in esse contenuti vengano continuamente riferiti da parte di certe applicazioni. Ciò comporta un sensibile aumento dei *fault di pagina*, con conseguente degrado delle prestazioni, in quanto una applicazione non riesce a tenere in memoria tutto il proprio insieme di lavoro (*working set*). Se invece la percentuale degli oggetti presenti in una pagina, e necessari ad un'applicazione, è molto alta, viene sprecato poco spazio di *buffer* con conseguente diminuzione del numero dei *fault di pagina*.

Va notato che le precedenti considerazioni valgono per SGBD sia con architettura centralizzata che cliente–servente, e più in generale per tutta la gerarchia di memorie coinvolta nel processo di elaborazione, comprese le risorse utilizzate per la trasmissione dei dati (bus, reti locali, ecc...). Ad esempio, supponiamo di avere una architettura cliente–servente su rete locale e

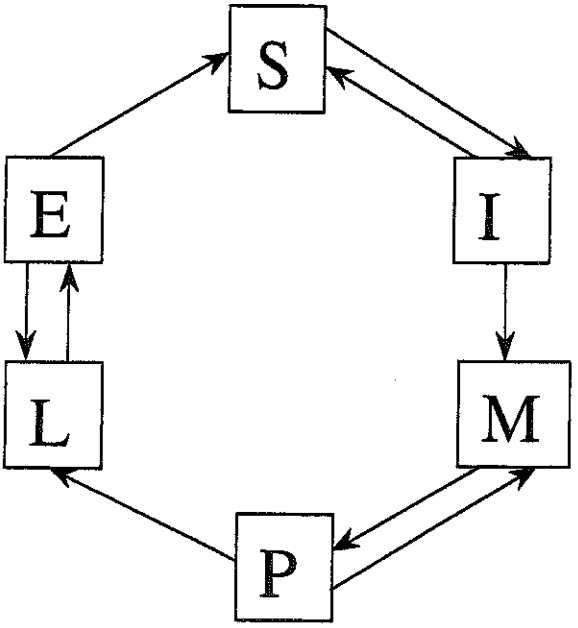


Figura 2: Associazioni tra gli oggetti nell'esempio *SIMPLE*

che i soliti tre oggetti siano residenti su differenti pagine del disco. In un contesto architetturale siffatto, potrebbe risultare vantaggiosa anche la presenza di un semplicissimo meccanismo di prefetching che permettesse di recuperare automaticamente tutte e tre le pagine fisiche e inviarle in un colpo solo al cliente. In questo caso, il vantaggio consisterebbe nell'attivare una sola comunicazione fra servente e cliente, poiché il costo di trasmettere tre pagine con un'unica comunicazione è solo leggermente più alto di quello di trasmettere una sola pagina alla volta (mentre è assai più basso di quello necessario a trasmettere tre pagine con tre comunicazioni punto a punto).

L'esempio precedente è molto intuitivo. Per illustrare che non è sempre così semplice, mostriamo un secondo esempio ripreso da [Tsangaris 91] e meglio noto come l'*esempio SIMPLE*.

Si consideri la base di dati formata da *sei* oggetti rispettivamente di identificatori *S, I, M, P, L* ed *E* che sono in associazione tra loro secondo il grafo di Figura 2. Consideriamo un'applicazione che ha il seguente *flusso di accesso*:

$(SI)^{99}(MP)^{99}(LE)^{99}$

Questo significa che viene acceduto prima l'oggetto *S* e di seguito *I*; successivamente l'applicazione passa da *I* ad *S* e viceversa per 98 volte. Di seguito viene acceduto *M* e nuovamente l'applicazione accede ripetutamente per 99 volte ad *M* e *P*. Infine accede ad *L* e passa per altre 99 volte da *L* ad *E* e viceversa. Nel caso in cui ogni *pagina fisica* contenga al massimo *tre* oggetti, una ragionevole scelta di clustering è la seguente:

[*S, I, M*], [*P, L, E*][2]

In questo modo viene minimizzato lo spazio occupato e il numero di riferimenti ad oggetti in pagine fisiche diverse.

Nel caso in cui il sistema abbia però una capacità di buffering limitata ad una sola pagina, l'esecuzione dell'applicazione precedente, secondo la scelta di clustering sopra, comporta 198 *fault di pagina*. Infatti, quando l'applicazione accede per la prima volta ad *S* provoca un *fault*. Questo non avviene per i successivi 98 accessi da *S* ad *I* e viceversa, e per l'accesso ad *M*. Nel momento in cui viene richiamato *P* si verifica il *secondo fault*: la pagina [*S, I, M*], presente nel *buffer* viene rimpiazzata da [*P, L, E*]. I successivi passaggi da *P* a *M* e da *M* a *P* provocano nuovi *fault* . Quindi eseguire il flusso $(MP)^{99}$ dopo $(SI)^{99}$ comporta (2x99)-1 *fault di pagina*. A seguito dell'ultimo accesso a *P* nessun altro *fault* si verifica nell'esecuzione di $(LE)^{99}$.

Una migliore scelta di memorizzazione degli oggetti è la seguente:

[*S, I,* _], [*M, P,* _] [*L, E,* _][3]

In tal modo l'esecuzione dell'applicazione comporta soltanto *tre fault di pagina* anche se parte dello spazio di ogni pagina viene sprecato.

Questi semplici esempi illustrano esaurientemente la complessità del problema del clustering. Questo problema diventa assai complicato nei SGBDOO in virtù della grande espressività del modello dei dati, che è quindi assai distante dalla rappresentazione fisica dei dati, delle diverse modalità di accesso agli oggetti da parte delle applicazioni, cioè navigazionale e associativo, e della complessità dell'architettura dei sistemi a oggetti, tipicamente cliente-servente.

---

[2] i limiti di ogni pagina vengono rappresentati mediante paretesi quadrate.
[3] '_' indica spazio non occupato all'interno della pagina.

3. Il clustering nei SGBDOO: approccio ingegneristico

In un *SGBDOO,* il *clustering* è utilizzato per ottimizzare sia accessi di tipo *navigazionale* che quelli *associativi.* A complicare il problema rispetto ad un sistema basato sul modello relazionale interviene anche l'espressività del modello ad oggetti che sappiamo essere tanto elevata da includere aspetti procedurali quali i metodi, oltre alla gestione dell'ereditarietà e degli *oggetti compositi* [Kim 89].

La scelta di una strategia di clustering all'interno di un *SGBDOO,* implica la valutazione dei seguenti aspetti:

- quali criteri utilizzare nell'eseguire il clustering;
- quando eseguire il clustering;
- informazioni disponibili in input all'algoritmo di clustering;
- influenza dell'architettura del SGBD sul clustering;
- granularità del clustering;

Sulla base dei punti precedenti si possono dare delle linee guida nella scelta i una strategia di clustering. La scelta effettuata deve poi essere valutata in termini di beneficio apportato al sistema che ne fa uso, e questo comporta la definizione di metodologie e tecniche per misurare le prestazioni di un SGBDOO.

3.1 Quali criteri utilizzare nell'eseguire il clustering

Come precedentemente accennato, in un *SGBDOO* l'utilizzo del clustering è relativo all'ottimizzazione di accessi di tipo *associativo* e di tipo *navigazionale.* Si può partire da qui per classificare i numerosi criteri utilizzati per effettuare il clustering.

- Ottimizzazione di *accessi associativi.*
  - ✓ *Clustering basato sul tipo.* Gli oggetti vengono raggruppati sulla base del loro tipo, decidendo ad esempio che tutte le istanze di una classe costituiscono un unico cluster. In presenza di una gerarchia di generalizzazione, in un cluster possono essere inserite anche le istanze relative alla *superclasse* e alle sue *sottoclassi.*

- ✓ *Clustering basato sui valori;* il raggruppamento avviene sulla base del valore di un attributo o di un gruppo di attributi. L'attributo il cui valore è utilizzato per il clustering viene detto *attributo di clustering*. Un particolare tipo di clustering basato sui valori è il *clustering basato sull'indice* che consiste nel costruire un indice sull'attributo scelto per eseguire il raggruppamento.

↳ Ottimizzazione di *accessi navigazionali.*

L'accesso navigazionale consiste nel percorrere i *riferimenti tra gli oggetti* (*inter object references*) rappresentanti i vari tipi di relazione. Allo scopo di ottimizzare le prestazioni alcuni sistemi permettono di creare cluster di oggetti in accordo ad una particolare *relazione* tramite cui gli oggetti sono associati. Un esempio di quanto detto è il *clustering basato sugli oggetti composti.*

- ✓ *Clustering basato sugli oggetti compositi.* Molte applicazioni richiedono la capacità di definire e manipolare, come singola *entità logica,* un insieme di oggetti correlati. L'insieme di *oggetti componenti* che formano una singola *entità logica* è detto *oggetto composito* [Kim 89]. Significa che ogni oggetto componente è "parte" dell'oggetto che lo riferisce, e non soltanto che è in semplice *associazione* con esso. La relazione che lega gli oggetti è detta *part-of.* Un *oggetto composito* presenta le seguenti caratteristiche: non ha cicli di associazioni *part-of,* ha un unico *oggetto radice* e al suo interno un oggetto componente può essere *parte* di più genitori. La cancellazione dalla base di dati dell'*oggetto radice* innesca la cancellazione di tutti gli *oggetti componenti.* In base a questo si può stabilire che gli oggetti appartenenti ad un *oggetto composito* appartengano allo stesso cluster. Questo modo di eseguire il clustering è particolarmente vantaggioso nel caso in cui le transazioni che accedono ad un oggetto implicano un successivo accesso alle sue componenti.
- ✓ *Clustering basato sul partizionamento del grafo degli oggetti.* Verrà trattato in maniera estesa nel prossimo paragrafo.

I modi di eseguire il clustering precedentemente descritti possono essere adottati da soli oppure congiuntamente ad altri. Ad esempio ORION ([Kim 90])

utilizza una strategia *basata sul tipo* congiuntamente a quella *basata sugli oggetti compositi.*

Oltre alle possibilità indicate sopra, occorre aggiungere che esiste anche il caso in cui è il *progettista della base di dati* e non il sistema in modo automatico a scegliere quale criterio utilizzare per il clustering. Ad esempio al momento della creazione di un oggetto x il progettista può *suggerire* al *SGBD*[4] di memorizzare l'oggetto *vicino* ad un oggetto y precedentemente creato. Questo approccio ha un grosso svantaggio: delega interamente ai progettisti la responsabilità delle scelte relative al clustering; dunque chi scrive una applicazione dovrebbe conoscere anche il comportamento delle altre applicazioni nei riguardi degli schemi dati.

3.2 Quando eseguire il clustering

L'istante in cui il clustering viene eseguito porta a distinguere due tipi di algoritmi: *statici* e *dinamici.* Tra questi due estremi esistono comunque molteplici soluzioni intermedie. Un esempio è dato in [Bancilhon 92].

3.2.1 Algoritmi statici

Nel caso *statico,* il clustering viene eseguito nel momento in cui gli oggetti vengono creati e non viene eseguita alcuna riorganizzazione della base di dati (*re-clustering*) a seguito della sua evoluzione. Le tecniche *statiche* portano ad una buona organizzazione iniziale della base di dati, ma hanno il difetto di produrre una caduta delle prestazioni nel tempo. In un ambito applicativo in cui sono frequenti le modifiche agli oggetti, questo può compromettere la validità dell'organizzazione fisica della base di dati. Per ovviare al problema alcuni sistemi eseguono periodiche riorganizzazioni *off-line* della base di dati per prevenire il degrado delle prestazioni del sistema (Encore [Hornick 87]).

3.2.2 Algoritmi dinamici

Le strategie di clustering *dinamiche* coinvolgono sia i *nuovi oggetti creati* da una applicazione che quelli già persistenti utilizzati nel corso dell'elaborazione. Questo significa che quando gli oggetti vengono trasferiti dalla memoria secondaria in quella principale, la pagina (o il blocco disco) in cui sono

[4] ma non è detto che il sistema ne possa tenere conto.

memorizzati inizialmente non è necessariamente la stessa nella quale si ritroveranno al termine dell'elaborazione. La riorganizzazione avviene così contemporaneamente al processo di clustering dei nuovi oggetti. Sempre in virtù della tipologia di accesso che si vuole ottimizzare, possiamo dire che le tecniche *dinamiche* sono particolarmente adatte ad ambiti applicativi in cui le modifiche agli oggetti hanno una frequenza dominante sugli accessi per la sola lettura. Si noti che il clustering dinamico comporta un costante overhead per l'esecuzione di transazioni che deve essere giustificato da un effettivo guadagno complessivo in termini di prestazioni del sistema.

3.2.3 Clustering statico/dinamico e identificatori di oggetti

La possibilità di *clustering dinamico*, cioè il trasferimento degli oggetti dalla pagina in cui sono stati inizialmente memorizzati ad un'altra, è fortemente influenzato dalla natura degli *identificatori degli oggetti* (OID).

Nel caso in cui il sistema utilizzi OID *logici*, essendo questi indipendenti dalla locazione fisica dell'oggetto, è più facile implementare strategie di clustering *dinamiche* poiché lo spostamento di un oggetto da una pagina ad un'altra lascia inalterati i riferimenti ad esso da parte di altri oggetti. La cosa è più complessa da realizzare nei sistemi con OID *fisici*. In questo caso l'utilizzo del clustering *dinamico* implica, a seguito dello spostamento di un oggetto, la modifica dell'OID e dunque di tutti i riferimenti a quell'oggetto contenuti in altri oggetti. Il problema può essere risolto facendo sì che il sistema mantenga per ogni riferimento ad oggetto anche i riferimenti inversi, oppure, più drasticamente, scandendo in maniera esaustiva la base di dati alla ricerca dei riferimenti da modificare. Entrambe le scelte sono improponibili in quanto comprometto inevitabilmente le prestazioni.

Si può dunque concludere che in SGBDOO, dove per ragioni di efficienza si utilizzino *OID fisici*[5] non è consigliato, a meno di costi proibitivi, scegliere strategie di clustering di natura dinamica.

[5] Che rendono più rapida l'operazione di navigazione dei riferimenti in quanto non necessitano di *tabelle di traduzione* come avviene per gli *OID logici*.

3.3 Input e parametri di controllo dell'algoritmo di clustering

Un aspetto fondamentale riguardante le varie strategie di clustering è costituito da quali dati vengono forniti in input all'algoritmo e da come questi dati vengono prodotti e memorizzati.

I dati di ingresso sono generalmente costituiti da *modelli di accesso agli oggetti* attraverso i quali si ricavano le informazioni sulla frequenza di accesso da un oggetto ad un altro in relazione con esso. Queste informazioni possono essere fornite esplicitamente dal *DBA*, oppure essere raccolte dal sistema in modo automatico sotto forma *di parametri statistici*. La raccolta di questi ultimi può avvenire attraverso un *monitoraggio dinamico* del sistema durante un periodo chiamato *fase di addestramento* (*training phase*) dell'algoritmo di clustering oppure in base ad una *analisi statica* applicata al codice dei metodi che produce un insieme di *cammini pesati* i quali costituiscono un modello sintattico della catena dei riferimenti attraversata dal metodo.

Bisogna anche tener presente che le informazioni che costituiscono i dati di ingresso dell'algoritmo di clustering, possono essere memorizzate a due livelli differenti: a *livello degli oggetti* e a *livello di tipo*. Nel primo caso la frequenza di accesso è memorizzata insieme all'identificatore dell'oggetto (sia esso logico o fisico).

L'overhead di questa soluzione è dato dal costo di tenere aggiornata la frequenza di accesso a un oggetto. Nel caso in cui le informazioni vengano memorizzate a livello di tipo (e perciò facenti parte dello schema dei dati), le interazioni tra oggetti sono derivate dal tipo stesso nel momento in cui si crea un nuovo oggetto.

Oltre alle informazioni di input, un altro fattore discriminante per gli algoritmi di clustering è costituito dai diversi *parametri di controllo* che influenzano le decisioni. Tali parametri possono essere classificati in *parametri di sistema* e *parametri della base di dati*. Oltre a questi, di cui parleremo nei paragrafi successivi, è possibile che l'algoritmo accetti anche parametri forniti direttamente dal *progettista della base di dati*.

3.3.1 Parametri di sistema

Sono strettamente collegati all'architettura del sistema e i più importanti sono: la *politica di gestione del buffer* e quella di *gestione delle pagine*. Per quanto

riguarda la gestione del buffer ci sono due tecniche principali: *inter transaction* e *intra transaction buffering*. Nel primo caso i dati presenti nel buffer vengono mantenuti tra una transazione e l'altra, mentre nell'intra transaction al termine di ogni transazione il buffer viene svuotato. La gestione delle pagine dipende, come accennato in precedenza, dal sottosistema di memorizzazione utilizzato. Se la pagina scelta dall'algoritmo di clustering per la memorizzazione di un oggetto è piena, si può eventualmente decidere di dividerla in due pagine o di cercarne una nuova libera possibilmente fisicamente adiacente alla precedente. La decisione deve essere fatta in modo tale da trovare un giusto compromesso tra il costo della divisione e quello della ricerca di una nuova pagina.

3.3.2 Parametri della base di dati

I parametri della base di dati possono essere in relazione a proprietà della base di dati quali: dimensione degli oggetti in ogni classe, numero delle istanze di ogni classe, grado di condivisione dei riferimenti tra oggetti, numero di riferimenti ad altri oggetti; oppure a proprietà relative al comportamento degli oggetti: rapporto accessi/modifiche, frequenza di invocazione dei metodi, frequenza di utilizzo di una associazione tra due oggetti.

3.4 Architettura dei SGBDOO e clustering

L'affermazione dei *SGBDOO* è andata di pari passo con quella delle architetture *cliente-servente*. In questo tipo di architettura le funzionalità del *SGBD* sono svolte da due processi distinti che possono, o risiedere sulla stessa macchina e comunicare per esempio attraverso memoria condivisa, oppure risiedere su macchine diverse e comunicare via rete. Il processo *servente* (*back-end*) è unico e svolge le funzionalità di memorizzazione, protezione e controllo degli accessi concorrenti ai dati; il processo *cliente* (*front-end*), che è replicato su più macchine, ospita le applicazioni che utilizzano la base di dati e il suo compito è quello di sottoporre le transazioni al servente, eventualmente traducendole dal linguaggio usato dell'utente, di ricevere dal servente i risultati e di organizzarli in forma opportuna per mostrarli all'utente.

Sistemi di questo tipo non possono dirsi *distribuiti* in senso stretto, in quanto i dati sono gestiti da un unico servente in maniera centralizzata e non vi sono dati replicati. Essi costituiscono, di fatto, un adattamento dell'architettura

centralizzata alla moderna concezione dei sistemi di elaborazione, visti come reti di calcolatori.

Per quanto riguarda le architetture *cliente-servente* per i SGBDOO, esse vengono classificate in base al tipo di informazioni scambiate tra cliente e servente. In base a questo, si possono individuare due alternative principali ben definite e per ognuna di queste un numero di varianti più o meno marcatamente distinguibili.

La prima alternativa, nota come architettura con *servente a oggetti* (*object server*), utilizza l'*oggetto* come unità di base per i trasferimenti nelle comunicazioni tra il cliente e il servente. In questo schema, il *servente* scambia con il cliente uno o più oggetti per messaggio e, nel caso in cui sia in grado di interpretare la loro semantica, il servente può eseguire anche i metodi. Esempi di questa architettura si possono trovare nel prototipo V1.0 di O_2 [Bancilhon 88], e in Orion 1 [Kim 90], nonchè in alcune versioni preliminari di GemStone [Copeland 84].

La seconda possibilità è costituita dall'architettura con *servente di pagine* (*page server*), nella quale l'unità di scambio è costituita dalla pagina di memoria secondaria. In questo caso il servente non è in grado di riconoscere e interpretare la semantica degli oggetti e perciò non è capace di eseguire i metodi. Questa architettura è stata utilizzata, ad esempio, nello sviluppo dei prototipi ObServer [Skarra 86] e Exodus [Carey 90].

Soffermandoci sulle architetture con *servente a oggetti* e con *servente di pagine*, analizziamo la loro influenza sul clustering. In un *servente a oggetti* il servente è in grado di comprendere e utilizzare il concetto di oggetto che costituisce l'unità di trasferimento nelle comunicazioni col cliente. Le scelte sul clustering vengono dunque effettuate dal servente che sarà dotato di un modulo appositamente progettato e dove è noto anche il concetto di pagina. Dunque quando una transazione eseguita dal cliente termina (*commit*), tutti gli oggetti persistenti devono essere restituiti al servente prima di eseguire l'algoritmo di clustering.

Nel caso il SGBDOO adotti una architettura basata sul *servente di pagine*, il concetto di oggetto è noto soltanto al livello del cliente che riceve dal servente le pagine della base di dati. Se le scelte di clustering sono appropriate ogni pagina conterrà gli oggetti necessari ad una applicazione senza richiedere

immediati trasferimenti addizionali. In questo caso le decisioni relative alla strategia di clustering sono a carico del cliente.

Alla luce di quanto detto sopra si può aggiungere che in un *servente di pagine* l'adozione di una strategia di clustering *statica* sia la scelta migliore. Questo perché una eventuale riorganizzazione della base di dati potrebbe coinvolgere pagine non presenti nel *working-set* del cliente al momento della riorganizzazione e comportare supplementari comunicazioni col servente.

3.5 Granularità del clustering

In genere un algoritmo di clustering produce come output quelli che possiamo chiamare *cluster logici* in quanto hanno potenzialmente una dimensione illimitata e non c'è garanzia che possano essere memorizzati all'interno dell'unità fisica di memoria secondaria a cui ha accesso il sistema (segmento o pagina disco).

Il problema è dunque quello di mappare i *cluster logici* in *cluster fisici* in modo da raggiungere il massimo profitto per ciò che riguarda il clustering. Generalmente ci sono due livelli di rappresentazione dei cluster fisici: *pagine fisiche* e *segmenti fisici*[6].

La prima alternativa è particolarmente vantaggiosa nel caso in cui i cluster logici prodotti dall'algoritmo siano piccoli. Questo è il caso, ad esempio, in cui la strategia di clustering sia *basata sugli oggetti compositi* o *sul valore*.

La seconda alternativa invece è utile nel caso di cluster logici relativamente grandi che occupano spazio disco costituito da più settori adiacenti o idealmente molto vicini tra loro. La corrispondenza tra ogni segmento e le pagine componenti viene mantenuta attraverso una tabella. Questa rappresentazione dei cluster fisici si adatta bene a sistemi dove la strategia di clustering è *basata sul tipo* e in cui è elevato il numero di istanze per ogni classe.

In ogni caso la scelta della granularità dipende sempre dalle caratteristiche del *sottosistema di memorizzazione* utilizzato dal SGBDOO e dalle caratteristiche di gestione della memoria secondaria del sistema operativo. Infatti se quest'ultimo non permette di specificare la *contiguità fisica* delle pagine da allocare, l'utilizzo dei *segmenti fisici* si risolve in una scelta di *pagine fisiche casualmente residenti sul disco.*

---

[6] Intesi come insiemi di *pagine fisiche* fisicamente adiacenti.

3.6 Scelta dell'algoritmo di clustering

Nella progettazione e realizzazione di un SGBDOO la scelta di una determinata strategia di clustering dipende dal contesto in cui questa deve operare e da molteplici fattori quali [Bertino 94]:

- *Proprietà strutturali degli oggetti.*
  Tra queste possiamo citare:
  - ✓ Dimensione media degli oggetti per ogni classe;
  - ✓ Numero medio di oggetti per ogni classe;
  - ✓ Grado di condivisione dei riferimenti tra oggetti;
  - ✓ Numero degli oggetti riferiti da un qualunque altro oggetto della base di dati;
- *Proprietà comportamentali degli oggetti.*
  Di queste citiamo:
  - ✓ Tipologia delle interrogazioni applicate ad ogni istanza di classe della base di dati;
  - ✓ Frequenza di esecuzione delle interrogazioni;
  - ✓ Strategia di esecuzione delle interrogazioni;
  - ✓ Numero di transizioni da un oggetto ad un altro durante una navigazione;
- *Obiettivi di prestazioni del sistema.* Questo consente di guidare la scelta discriminando tra tecniche di clustering di complessità media oppure (nel caso in cui l'indice di prestazioni del sistema debba essere elevato) tecniche di maggiore complessità ed efficacia (come il *clustering dinamico*) ponendo in secondo piano l'overhead che il loro utilizzo comporta.

3.6.1 Linee guida nella scelta di dell'algoritmo di clustering

Nella scelta di un algoritmo di clustering il primo passo da effettuare è quello analizzare le caratteristiche del *SGBD* che lo deve supportare in modo di decidere se adottare un strategia *statica* o *dinamica* e stabilire quale influenza avrà l'architettura nelle scelte successive. Un fattore da considerare ad esempio è dato dal modo (*logico* o *fisico*) in cui il *SGBD* rappresenta gli OID. Altri fattori che consentono di discriminare tra strategie statiche e dinamiche rientrano in quelle che abbiamo chiamato *proprietà comportamentali degli oggetti* e tra questi in particolare il rapporto *accessi/modifiche* sulla base di dati. Il clustering dinamico ha un costo di gestione, a tempo di esecuzione, che deve essere giustificato in

quanto aumenta il tempo di risposta in fase di modifica ma impedisce il degrado nel tempo dei tempi di accesso. Di conseguenza una strategia di clustering dinamica migliora il tempo complessivo di risposta soltanto in ambienti dove il rapporto *accessi/modifiche* sia elevato ossia c'è una netta predominanza delle *accessi* sulle *modifiche*. Tutte queste considerazioni vanno fatte anche alla luce delle caratteristiche del *sottosistema di memorizzazione* utilizzato e del *sistema operativo* a disposizione. Una volta fissati gli *obiettivi di prestazione* del *SGBDOO* si può adottare un *criterio in base al quali eseguire il clustering*. Le considerazioni che si possono fare in questo caso sono le seguenti.

Il *clustering basato sul tipo* è una buona scelta quando le *proprietà comportamentali* indicano che vengono eseguiti ripetuti accessi ad oggetti della stessa classe; questo accade in presenza di interrogazioni che hanno come fattore di selezione predicati su intervalli e ricerche trasversali. La sua efficacia aumenta nel caso in cui si operi con oggetti la cui dimensione media è relativamente piccola rispetto alla dimensione della *pagina fisica* scelta come unità di clustering.

Le strategie di clustering *basate sui valori* sono utilizzate in ambienti in cui le interrogazioni sono costituite da combinazioni booleane di predicati semplici (predicati di uguaglianza o basati su intervalli, applicati su oggetti della stessa classe). L'efficacia è maggiore quando il *grado di condivisione dei riferimenti tra oggetti* (*proprietà strutturale degli oggetti*) è alto.

Il *clustering basato sugli oggetti compositi* o più genericamente i criteri che tendono a rendere più efficienti gli *accessi navigazionali*, sono utilizzati quando le *proprietà comportamentali* degli oggetti indicano frequenti accessi che utilizzano le gerarchie di oggetti compositi e in cui la maggioranza degli oggetti componenti è necessaria all'elaborazione. Come nel caso del *clustering basato sul tipo*, anche questa strategia risulta dare maggiori benefici nel caso in cui si operi con oggetti la cui dimensione media è relativamente piccola rispetto alla dimensione della *pagina fisica* scelta come unità di clustering; in tal caso si riesce a memorizzare tutto un *oggetto composito* in una *pagina fisica*.

3.7 Valutazione delle prestazioni

L'efficacia di un algoritmo di clustering si misura valutando l'incremento delle prestazioni del SGBD che lo utilizza. Ci sono comunque, come vedremo,

metodologie di valutazione che permettono di rendere lo studio dell'algoritmo di clustering indipendente dalle prestazioni sistema.

La misura delle prestazioni di un sistema può essere effettuata utilizzando un *benchmark* che possiamo definire come un insieme standard di test a cui viene sottoposto un sistema, sia esso solo hardware che software, allo scopo di verificarne l'efficienza e la reale utilizzabilità, in rapporto alla risoluzione di una determinata categoria di problemi per cui il sistema è stato costruito.

I *benchmark* per i *SGBD* si basano generalmente su una *base di dati* specifica sulla quale vengono effettuate una serie di *operazioni rappresentative di una determinata categoria di applicazioni.* In tal modo le misure riportate nell'esecuzione delle operazioni mostrano come il SGBD reagisce alle tipiche operazioni del benchmark su quella base di dati.

In letteratura si trovano descritti molti benchmark espressamente progettati per *SGBDOO* e tra questi possiamo ricordare: Syntethic Benchmark [Kim 90], HyperModel Benchmark [Anderson 90], OO1 [Cattel 92], OO7 [Carey 93], e CluB-0 Benchmark [Bancilhon 92].

Decidere su quali aspetti porre l'attenzione nel misurare le prestazioni di un *SGBDOO* non è stato un problema di semplice soluzione. Sono stati selezionati come interessanti i seguenti argomenti [Joseph 89]:

- velocità nel navigare attraverso la base di dati;
- numero di transazioni eseguite nell'unità di tempo;
- tempo di risposta ad una interrogazione;
- tempo di memorizzazione e recupero dello stato di oggetti complessi sulla base della loro identità.

Altri punti critici riguardanti le prestazioni di un *SGBDOO* possono essere costituiti da:

- meccanismi di astrazione forniti dal SGBDOO;
- velocità di I/O dei dispositivi su cui è memorizzata la base di dati;
- tempo speso nella trasformazione da formato di memoria temporanea a formato di memoria permanente e viceversa;
- de-referenziamento dei puntatori agli oggetti persistenti;
- comunicazioni fra processi.

La maggior parte dei benchmark esistenti focalizza l'attenzione su alcuni dei precedenti aspetti, tralasciandone altri ritenuti meno importanti dal punto di

vista della tipologia di applicazione che utilizzerà il sistema testato. Questo comporta che sistemi ottimi per un benchmark risultino mediocri per altri e quindi meno adatti a supportare le applicazioni di cui quel benchmark è rappresentante. La cosa si riflette anche sugli algoritmi di clustering che saranno giudicati buoni o cattivi in base al contesto applicativo, alle caratteristiche del sistema in cui operano e alle variabili considerate dal benchmark e al loro legame diretto o indiretto con la strategia di clustering utilizzata.

Per questo molti progettisti di *SGBDOO* per testare i sistemi utilizzano *modelli di simulazione,* quali ad esempio [Tsangaris 91] [Tsangaris 92] [Chang 89] [Cheng 91] [Darmont 95] e [He 93]. Essi permettono di misurare più specificatamente le variazioni di prestazione dovute ad esempio all'utilizzo di un algoritmo di clustering rispetto ad un altro. Tsangaris e Naughton in [Tsangaris 92] propongono un duplice metodo di valutazione delle prestazioni che consiste nell'eseguire simulazioni utilizzando la base di dati introdotta nel benchmark CluB-0; il loro modello è sviluppato per un ambiente *multi-client* con l'utilizzo di *inter-transaction buffering.*

Chang e Katz in [Chang 89] mostrano invece un modello di simulazione che utilizza diversi *parametri di controllo* per testare la riduzione delle operazioni di I/O durante la *riorganizzazione off-line* effettuata utilizzando un algoritmo greedy.

Cheng e Hurson ([Cheng 91]) hanno proposto invece due modelli di simulazione per valutare i loro studi. Il primo modello riguarda il problema della gestione della modifica degli oggetti attraverso uno schema di clustering dinamico. Il secondo modello serve invece a testare il loro *schema di clustering a livelli multipli* utilizzato in contesti dove sono presenti *associazioni multiple* tra gli oggetti.

In [Darmont 95] l'approccio suggerito, propone invece di utilizzare una metodologia di modellazione che permette di confrontare le prestazioni di diversi *SGBDOO* e in particolar modo delle strategie di clustering; il modello ottenuto viene poi utilizzato per effettuare simulazioni. Il principale vantaggio dell'approccio di Darmont rispetto ai benchmark è dovuto alla possibilità di confrontare gli algoritmi di clustering in modo del tutto indipendente da ogni variabile di ambiente (ad es. la strategia di buffering e la gestione della memoria secondaria) associata al *SGBDOO* che implementa l'algoritmo di clustering in esame. Inoltre tale metodologia consente lo studio a priori di un

Distanza media di posizionamento = numero totale di pagine della base di dati/numero di pagine lette

Tasso di Fault = numero totale di *page fault*/numero di oggetti richiesti durante una operazione

Efficienza = *NRP/ONRP*
dove *NRP* rappresenta il numero delle pagine lette e *ORNP* il numero ottimale di pagine da leggere, calcolato conoscendo la dimensione degli oggetti e lo spazio che essi occupano sul disco

Fattore di Perdita = *(NRP-ONRP)/NRP*
che rappresenta la percentuale delle operazioni di I/O in eccesso rispetto al numero ottimale

Dimensione media dell'insieme di lavoro

Fattore di Espansione (EF)
misura la capacità di raggruppamento dei dati dell'algoritmo di clustering

$$EF = \sum (P(Q_i) \times EF(Q_i))$$

dove:
$P(Q_i)$ = probabilità di esecuzione della query Q_i;

$$EF(Q_i) = \frac{N(Q_i)}{\lceil |Q_i|/L \rceil}$$

$N(Q_i)$ = numero di pagine necessarie per l'esecuzione della query Q_i;
L = dimensione di un cluster;
$|Q_i|$ = numero di oggetti necessari alla query Q_i;

Figura 3: Indici di prestazioni più frequentemente utilizzati

algoritmo di clustering (nell'articolo di Darmont viene esaminato l'algoritmo *CK* [Chang 90]) prima della sua implementazione in un *SGBDOO*.

Alla luce di quanto detto si vede dunque come valutare un algoritmo di clustering sia un compito difficile in quanto metodologie diverse possono condurre a risultati contrastanti in virtù del fatto che il comportamento dei vari algoritmi testati è influenzato da molte variabili (tra i quali anche i *parametri di controllo* discussi in 3.3) e differisce in accordo agli obiettivi di performance del *SGBD*, alla struttura dei suoi oggetti e alla natura delle applicazioni per cui il *SGBDOO* è stato progettato.

Nella Figura 3 vengono riportati gli indici più frequentemente utilizzati nella valutazione delle prestazioni di un SGBDOO o di uno specifico algoritmo di clustering così come presi da [Bertino 94].

4. Il clustering nei SGBDOO: approccio matematico

Il problema del clustering può essere generalmente ricondotto ad un problema di ottimizzazione su un *grafo* la cui soluzione consiste nel ricavare da questo un certo numero di *sottografi* (le *partizioni*), non connessi tra loro, rappresentanti i *cluster logici*.

L'input del problema è costituito dal *grafo degli oggetti (OG)* in cui i *nodi* rappresentano gli *oggetti* e gli *archi* le *associazioni* tra essi (*inter object references*). L'algoritmo di clustering ha lo scopo di partizionare *OG* assegnando gli oggetti alle *pagine fisiche* della base di dati. Più spesso un algoritmo di clustering invece che sul *grafo degli oggetti* opera su un grafo più specifico detto *grafo di clustering (CG)*. Questo deriva direttamente da *OG* con la differenza che sia i *nodi* che gli *archi* sono *pesati*: i *pesi* associati ai *nodi* rappresentano la *dimensione degli oggetti*, mentre i *pesi* associati agli *archi* denotano la frequenza di accesso di quella *associazione* da parte di una transazione. Per un dato partizionamento del *CG*, il *peso* di tutti gli *archi* che consentono di passare da una partizione ad un'altra è detto *costo esterno del partizionamento*. La soluzione al problema del *clustering* consiste nel trovare un partizionamento di *CG* in modo tale che la somma delle dimensioni degli oggetti appartenenti ad ogni partizione sia *minore* o *uguale* alla dimensione di una *pagina fisica* e il *costo esterno del partizionamento* sia *minimizzato*.

Ciò premesso, ci sono dunque due dimensioni ortogonali lungo le quali classificare gli algoritmi di clustering: (a) la *determinazione dei pesi* associati agli archi del *CG*, che può essere fatta *staticamente* (ad esempio attraverso l'analisi statica delle transazioni) o *dinamicamente* (attraverso il monitoraggio del sistema) e (b) il modo in cui gli oggetti del *CG* vengono associati alle *pagine fisiche*. Nel proseguo della sezione ci concentreremo su questo seconda dimensione che riteniamo più interessante ai nostri scopi e in base alla quale gli algoritmi di clustering si dividono in:

- algoritmi *basati su sequenze*,
- algoritmi *basati su partizioni*.

Gli algoritmi di clustering classificabili come *basati su sequenze* partizionano il *CG* applicando una *procedura di visita* (*graph traversal*) che trasforma il grafo in

una *sequenza lineare ordinata* di oggetti assegnati poi in tale ordine a *pagine fisiche.*

Nel caso di algoritmi *basati su partizioni,* invece, per il partizionamento del *CG* si utilizzano i classici *algoritmi di partizionamento grafi.* Gli oggetti di ogni sottografo ottenuto vengono poi memorizzati in una *pagina fisica.*

Tsangaris e Naughton in [Tsangaris 91] e [Tsangaris 92] hanno dimostrato che gli algoritmi *basati su partizioni* danno risultati migliori in termini di prestazione rispetto a quelli *basati su sequenze* ma sono penalizzati rispetto a questi ultimi da una *complessità in tempo* decisamente maggiore che li rende in molti casi di fatto inutilizzabili.

4.1 Algoritmi basati su sequenze

L'idea di base di un algoritmo di clustering *basati su sequenze* può essere riassunta in due passi principali:

1. *si ottiene dal CG una* sequenza di oggetti *detta sequenza di clustering;*
2. gli oggetti sono memorizzati in *pagine fisiche* in accordo alla sequenza.

Mentre il secondo passo è ovvio, esistono molte soluzioni per risolvere il problema della generazione di una *sequenza di clustering.* Queste tuttavia possono essere tutte descritte mediante un algoritmo che consiste a sua volta di altri due passi, indicati come *PreSort* e *Traversal* e che, utilizzando la notazione con la quale nel sistema operativo UNIX si denotano le *pipe*[7], può essere così descritto:

PreSort | *Traversal*

Nella prima fase gli oggetti da raggruppare sono *ordinati* applicando un procedimento di *PreSort;* quanto ottenuto costituisce l'input per la seconda fase nella quale si utilizza una procedura detta *Traversal.* Quest'ultima passa in rassegna tutti gli oggetti della sequenza prodotta dal *PreSort* ed è parametrica rispetto al primo oggetto non visitato della sequenza; a partire da questo il suo scopo è quello di attraversare tutti gli oggetti raggiungibili. Questo procedimento viene ripetuto fin quando non sono stati visitati tutti gli oggetti

[7] Una *pipe* prende gli elementi prodotti in *output* dalla prima componente - quella a sinistra del simbolo '|'- e li passa come *input* alla seconda componente - quella a destra del simbolo '|' - nello stesso ordine in cui sono stati prodotti.

del *CG*. Il risultato di questa seconda fase costituisce quella che precedentemente abbiamo chiamato *sequenza di clustering* e in accordo alla quale si memorizzano gli oggetti in *pagine fisiche*. Spesso gli oggetti sono memorizzati nell'istante stesso in cui vengono visitati; dunque la *Traversal* e la fase di memorizzazione fisica sono spesso fuse tra loro.

Affinché la strategia di clustering porti a risultati soddisfacenti, la *Traversal* deve operare sulla *sequenza di clustering* con un comportamento il più possibile simile a quello delle applicazioni attualmente eseguite sulla base di dati. Per comprendere al meglio questo fatto vediamo un esempio. Consideriamo una estensione della base di dati definita in accordo al seguente schema C++:

```
class PuntoGriglia {
private:
      PuntoGriglia x;
      PuntoGriglia y;
      PuntoGriglia z;
public:
      PuntoGriglia (x=y=z=NULL;);

      PuntoGriglia xMove(int n) {
            PuntoGriglia current;
            current = this;
            while ((current.x != NULL) && (n>0)) {
                  current = current.x;
                  n-= 1;
            }
            return current;
            };

      PuntoGriglia x_Move() {
            PuntoGriglia current;
            current = this;
            while (current.x != NULL) {
                  current = current.x;
            }
            return current;
      };

      PuntoGriglia y_Move() {
            PuntoGriglia current;
            current = this;
            while (current.y != NULL) {
                  current = current.y;
            }
            return current;
      };
};
```

L'estensione della base di dati in esame viene mostrata in Figura 4 e consiste di un certo numero di oggetti di tipo **PuntoGriglia** che formano una *griglia*

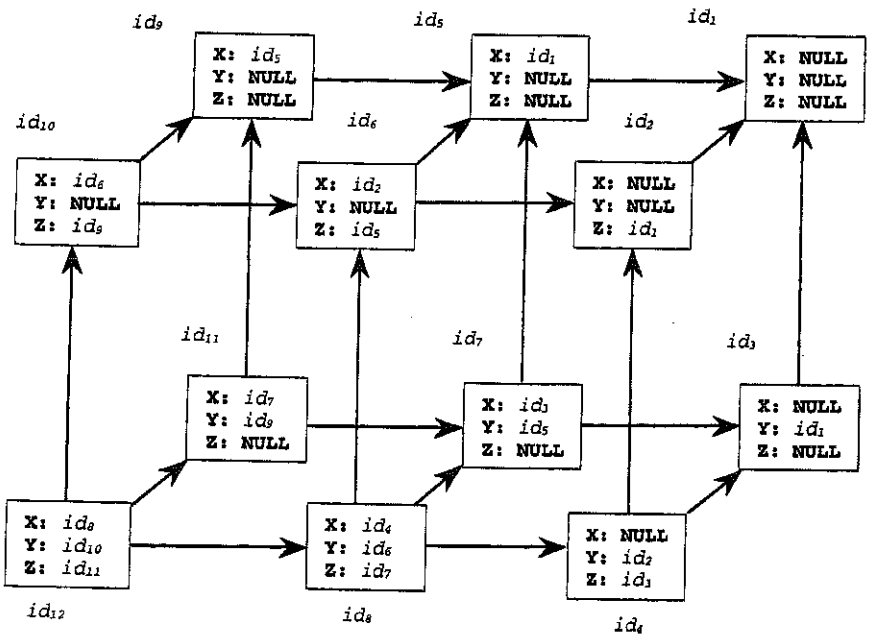


Figura 4: Estensione della base di dati PuntoGriglia

nello *spazio tridimensionale*. Il metodo **xMove(n)** permette ad un oggetto di raggiungerne un altro che si trova sulla griglia distante da esso **n**-oggetti in direzione **x**. I metodi **x_Move()** e **y_Move()** permettono di muovere un oggetto sulla griglia (rispettivamente nelle direzioni **x** e **y**) fino a che è possibile (cioè fino a che ci sono riferimenti in avanti diversi da **NULL**).

Per illustrare l'impatto della fase di *Traversal* facciamo le seguenti assunzioni:

- l'applicazione presa in considerazione è la id_9.*xMove(2)* che visita nell'ordine gli oggetti id_9, id_5 e id_1.
- ogni *pagina fisica* può contenere al massimo *tre* oggetti di tipo **PuntoGriglia**;
- l'ordine prodotto dalla fase di *PreSort* è: id_{12}, id_{11}, ..., id_1. Questo risultato del *PreSort*, che nel seguito indicheremo come cr^{-1}, sarà l'ordine inverso a quello nel quale gli oggetti sono stati creati (che indicheremo con cr).

Vediamo ora cosa comporta utilizzare come procedure per la *Traversal* rispettivamente i metodi **x_Move()** e **y_Move()**.

Utilizzando la seguente strategia:

cr^{-1} | **y_Move()**,

in base alle assunzioni precedentemente fatte, si ottengono i seguenti *cluster fisici* (le parentesi quadre delimitano la *pagina fisica*):

$[id_{12}, id_{10}, id_{11}], [id_9, id_8, id_6], [id_7, id_5, id_4], [id_2, id_3, id_1]$

Considerando l'esecuzione dell'applicazione $id_9.xMove(2)$, questo risultato comporta *tre fault di pagina.*

Adottando invece la strategia:

cr^{-1} | **x_Move()**,

si ottengono i seguenti *cluster fisici*:

$[id_{12}, id_8, id_4], [id_{11}, id_7, id_3], [id_{10}, id_6, id_2], [id_9, id_5, id_1]$

che comportano un *solo fault di pagina* durante l'esecuzione di $id_9.xMove(2)$. Come ci aspettavamo è dunque la *Traversal* più simile all'applicazione in esame a produrre i migliori risultati (minimizzare il numero di *fault di pagina*).

Analizziamo ora con un secondo esempio l'impatto della fase di *PreSort* che non è così ovvio come abbiamo visto per la *Traversal*. Anche in questo caso, come per il precedente, assumiamo che lo scopo della strategia di clustering sia quello di ottimizzare l'esecuzione dell'applicazione $id_9.xMove(2)$. La *Traversal* scelta in questo caso è la **x_Move()** e verrà utilizzata per confrontare i *PreSort*: *cr* e *cr*$^{-1}$.

Come abbiamo visto in precedenza l'utilizzo della strategia di clustering:

cr^{-1} | **x_Move()**,

comporta i seguenti cluster fisici:

$[id_{12}, id_8, id_4], [id_{11}, id_7, id_3], [id_{10}, id_6, id_2], [id_9, id_5, id_1]$

ed un *unico fault di pagina* nell'esecuzione della $id_9.xMove(2)$. Se si utilizza invece la strategia:

cr | **x_Move()**,

si otterranno i seguenti cluster fisici:

$[id_1, id_2, id_3], [id_4, id_5, id_6], [id_7, id_8, id_9], [id_{10}, id_{11}, id_{12}]$

e di conseguenza tre *fault di pagina*. Si vede dunque che, nonostante sia stata utilizzata la *Traversal* che nel caso precedente comportava un unico *fault di pagina*, la scelta della strategia di *PreSort cr* ha comportato un aumento dei *fault* rispetto a cr^{-1}. Questo significa che gli oggetti sono già stati posizionati dal *PreSort* prima che la *Traversal* percorra il grafo. Infatti le traiettorie seguite dalla *Traversal* `x_Move()` a partire dall'ordine costituito da *cr* prevengono di fatto ogni reale navigazione del grafo di Figura 4. Questo perchè il primo oggetto che deve essere passato alla *Traversal* (e che in seguito chiameremo *entry point*) secondo l'ordine stabilito da *cr* rappresenta un *nodo pozzo* del grafo. Dunque, la scelta di una buona strategia di *PreSort* è basata sulla ricerca un *entry point* nel *grafo degli oggetti* che permetta alla *Traversal* di operare correttamente.

Dal punto di vista del *SGBD* la più semplice strategia di *PreSort* può essere suggerita dal *progettista della base di dati*. Molto comune è la scelta di un *ordinamento basato sul tipo* ([Hornick 87] [Stamos 84] e [Benzaken 90]) che comporta però un *ordinamento parziale* in quanto gli oggetti dello stesso tipo non posso essere ordinati con questo criterio. A questa situazione si pone rimedio ordinando gli oggetti dello stesso tipo sulla base del *valore di un attributo*.

Un ulteriore modo di eseguire il *PreSort* è considerare l'ordine temporale in cui gli oggetti sono stati creati (che abbiamo indicato con *cr*). Questo si rivela molto spesso una buona scelta in quanto gli oggetti creati in istanti di tempo vicini sono spesso utilizzati congiuntamente da una applicazione.

Abbiamo visto inoltre come l'*entry point* per la *Traversal*, che dipende dal *PreSort*, influenza le sorti del procedimento di clustering. Nel caso precedente l'utilizzo di *cr* si è rivelato una scelta non felice proprio perché comportava un *entry point* che era un *nodo pozzo* del *grafo degli oggetti*. Per ovviare a questo problema si può specificare esplicitamente un *entry point* o un *insieme di punti*.

Altri tipi di *PreSort* possono essere basati su *contatori di riferimento statici* (*SRC*) o *dinamici* (*DRC*).

L'*SRC* di un oggetto è il *numero di riferimenti verso quell'oggetto* ossia il *grado entrante* del nodo rappresentante quell'oggetto nel *grafo degli oggetti*. Ad esempio, osservando la Figura 4, id_1 ha un *SRC* pari a *tre* mentre quello di id_{11} è pari ad *uno*. Numerosi *test* hanno rivelato che ordinare gli oggetti per *SRC crescente* ha un impatto positivo maggiore sulla strategia rispetto ad un ordinamento *decrescente*. Questo è giustificato dal fatto che gli oggetti non riferiti da altri spesso costituiscono gli *entry point* del grafo degli oggetti (ad

esempio in Figura 4 l'oggetto id_{12}). È da notare che ordinare rispetto allo *SRC* può dar luogo ad un *ordinamento parziale* in quanto più oggetti possono avere lo stesso *SRC*.

Il *DRC* di un oggetto tiene invece traccia del numero di volte che l'oggetto è stato acceduto da una applicazione. Un esempio di *PreSort* basato su *DRC* si trova in [Hudson 89].

La *Traversal* come abbiamo detto in precedenza è una operazione che riceve come argomento un oggetto ed ha il seguente comportamento: se l'oggetto ricevuto in *input* non è stato ancora memorizzato e c'è spazio nella *pagina fisica* correntemente in esame, allora viene memorizzato in essa. Se *non* è stato ancora memorizzato ma *non* c'è spazio nella pagina in esame ne viene scelta una nuova in cui avviene la memorizzazione. A questo punto viene scelto un nuovo oggetto che di solito è uno di quelli riferiti da un altro oggetto già visitato nella corrente esecuzione della *Traversal*. Una volta che questo oggetto è stato memorizzato ne viene scelto uno nuovo. Se la scelta non è possibile l'esecuzione della *Traversal* si conclude e se ne fa partire un'altra; il procedimento si ripete fin quando tutti gli oggetti della sequenza prodotta dalla *PreSort* sono stati elaborati.

Per la *Traversal* le strategie più frequentemente utilizzate sono *tre*:

- *Depth-First Traversal*: si visita il nodo di partenza (*entry point*) e successivamente si richiama la procedura ricorsivamente su tutti i suoi figli ordinandoli in base ad un qualche criterio. Esempi d'uso si possono trovare in [Stamos 84] e in [Banerjee 88].
- *Breadth-First Traversal*: a partire dall'*entry point* si visitano tutti i figli dell'ultimo nodo visitato. Esempi d'uso in [Stamos 84] e in [Banerjee 88].
- *Best-First Traversal*: questa *Traversal* è la più difficile da descrivere in quanto il suo comportamento dipende in ogni istante dagli oggetti già memorizzati nella base di dati. Ad ogni passo, tutti gli oggetti riferiti da altri già memorizzati sono ordinati in accordo ai pesi dei rispettivi archi che li riferiscono. Esempi d'uso sono in [Tsangaris 91], [Gerlhof 92] e in [Hudson 89].

4.2 Algoritmi basati su partizioni

Gli algoritmi di clustering *basati su partizioni* non sono altro che algoritmi di partizionamento grafi applicati al *grafo degli oggetti* opportunamente pesato. In

letteratura questi algoritmi sono apparsi per la prima volta nel lavoro di Tsangaris e Naughton ([Tsangaris 91] e [Tsangaris 92]) dove veniva utilizzato l'algoritmo *KL* [Kernighan 70]. I risultati ottenuti dall'algoritmo in questo caso, si sono rivelati superiori a quelli ottenuti utilizzando strategie *basate su sequenze* anche se il benchmark utilizzato non rispecchiava di fatto una situazione reale; infatti la base di dati era piuttosto piccola e gli oggetti tutti di dimensioni uguali e anche essi piccoli. Di fatto, come abbiamo precedentemente accennato, tali algoritmi sono scarsamente utilizzati nei SGBD rispetto a quelli *basati su sequenze* in quanto la loro complessità in tempo riduce i benefici apportati in termini di aumento delle prestazioni. In accordo con [Shahookar 91] gli *algoritmi per il partizionamento dei grafi* posso essere divisi in due categorie:

- *algoritmi costruttivi,* che preso in input il solo grafo degli oggetti producono un partizionamento;
- *algoritmi iterativi,* che preso come input il grafo degli oggetti ed un suo iniziale partizionamento, cercano soluzioni migliori procedendo per tentativi.

Solitamente gli algoritmi del secondo tipo producono risultati migliori rispetto ai primi ma hanno rispetto ad essi un tempo di esecuzione di gran lunga superiore. Nel seguito accenneremo ad alcuni degli algoritmi di partizionamento grafi noti in letteratura.

Algoritmi Iterativi

L'euristica Kernighan-Lin. L'algoritmo *KL* è stato progettato per applicazioni di tipo VLSI. Il suo funzionamento consiste nel partire da un arbitrario partizionamento del grafo degli oggetti e di migliorarlo scambiando gli oggetti tra una coppia di partizioni correntemente considerata. Per favorire gli scambi, gli oggetti devono essere di dimensione non molto grande e non molto diversa tra loro. Questa assunzione purtroppo è vera per applicazioni VLSI ma non per generiche applicazioni che lavorano su basi di dati orientate a oggetti.

L'euristica Fiduccia-Mattheyses. Questo algoritmo di partizionamento grafi (*FM*) (riportato in [Fidducia 82]) è stato derivato dall'algoritmo *KL* modificandolo per tener conto di partizioni non bilanciate e di oggetti di dimensioni non uniforme. Infatti non c'è più uno *scambio* di oggetti tra una coppia di partizioni ma una *migrazione* di oggetti da una partizione ad un'altra.

Questo algoritmo è anche in grado di adattare il numero di oggetti in ogni partizione e il numero delle partizioni (se tutti gli oggetti vengono spostati da una partizione e questa risulta vuota, l'algoritmo la cancella). *KL* al contrario non è in grado di modificare il numero di oggetti per partizione nè il numero iniziale delle partizioni che gli vengono passate.

Partizionamento gerarchico. L'algoritmo di partizionamento gerarchico (*HP*) consiste in una combinazione degli algoritmi *KL* e *FM*. Il funzionamento dell'algoritmo è il seguente: tutti gli oggetti sono assegnati ad una partizione la cui dimensione è $P = 2^x \times PageSize$ dove x è il più piccolo intero tale che la dimensione totale di tutti gli oggetti sia *minore* o uguale a P. Questa partizione viene divisa in due in modo tale che gli oggetti siano equamente distribuiti[8] tra le due partizioni. Alle partizioni risultanti viene applicato l'algoritmo *FM* o *KL*. Questo metodo viene applicato ricorsivamente fino a quando la dimensione di ogni partizione è inferiore a quella di una pagina fisica. L'idea di questo algoritmo è partita dal lavoro di Breuer [Breuer 77].

Algoritmi Costruttivi

Partizionamento ottimo di Alberi. In [Lukes 74] viene presentato un algoritmo con tempo di esecuzione pseudopolinomiale per il calcolo di un partizionamento ottimo dei nodi di un albero. La complessità dell'algoritmo è $o(n \times B^2)$ dove n è il numero di nodi dell'albero e B è il massimo numero di oggetti per pagina.

L'algortimo Greedy Graph Partitioning *(GGP).* Questo algoritmo è stato proposto per la prima volta in [Gerlhof 92]. Gli algoritmi greedy riescono a trovare in maniera efficiente buone soluzioni per i problemi di ottimizzazione come il partizionamento di un grafo. L'algoritmo *GGP* è basato su una semplice euristica greedy derivata dall'algortimo di Kruskal [Kruskal 56] per il calcolo del *minimo albero di copertura di un grafo*. L'algoritmo opera nel seguente modo: tutte le partizioni iniziali contengono un solo oggetto e tutte le partizioni vengono inserite in una *lista di partizioni* (*PartList*). Per tutti gli oggetti o_1 e o_2 connessi da un arco pesato $w_{1,2}$ nel grafo degli oggetti, viene creata la tripla $[o_1, o_2, w_{1,2}]$ che viene inserita nella *lista degli archi* (*EdgeList*). Vengono poi visitate tutte le triple in *EdgeList* nell'ordine di peso decrescente. Sia $[o_1, o_2, w_{1,2}]$ la tripla

[8] Per quanto riguarda le *dimensioni*.

correntemente in esame e P_1 e P_2 le partizioni a cui rispettivamente o_1 e o_2 sono assegnati. Se P_1 è diversa da P_2 e se la dimensione totale degli oggetti contenuti in entrambe le partizioni è minore della dimensione di una pagina, le due partizioni vengono fuse[9]. Altrimenti l'arco viene scartato e le partizioni rimangono invariate. Se e è il numero di archi del grafo, la complessità dell'algoritmo è $o(e \log e)$; ossia il fattore dominante è il costo per l'ordinamento della *lista degli archi*. I benchmark eseguiti su questo algoritmo hanno dimostrato che esso è il migliore nel *rapporto qualità del clustering, tempo di esecuzione*. In [Gerlhof 93] viene presentata una modifica dell'algoritmo *GGP* (denominato *GPP with look-ahead*) che ne migliora ulteriormente l'efficacia.

5. Conclusioni

Nelle precedenti sezioni abbiamo approfondito molti degli aspetti legati al clustering nei *SGBDOO*. Si è evidenziato come la difficoltà del problema risulta superiore rispetto ai sistemi tradizionali a causa della grande espressività del modello dei dati, delle diverse modalità con cui gli oggetti possono essere acceduti e della complessità dell'architettura.

Il problema è stato poi analizzato sulla base di due approcci che emergono maggiormente in letteratura: quello *ingegneristico* e quello *matematico*. Nel primo caso abbiamo discusso sui diversi criteri con cui può essere eseguito il clustering e l'influenza che le caratteristiche del *SGBD* hanno nella scelta della strategia da adottare. Abbiamo analizzato quali informazioni servono generalmente agli algoritmi di clustering per operare correttamente e accennato a come se ne valuti l'efficacia attraverso i *benchmark* e le *simulazioni*.

Per quanto riguarda invece l'approccio *matematico*, che riconduce il problema del clustering al più classico problema di trovare la partizione ottima di un grafo, le soluzioni presenti in letteratura sono state classificate in base al modo in cui si assegnano gli oggetti (nodi del grafo) a pagine fisiche della base di dati. Questo ha portato a distinguere due classi di algoritmo: *algoritmi basati su sequenze* e *algoritmi basati su partizioni*. Per entrambe queste classi abbiamo approfondito gli aspetti principali.

Alcune delle problematiche discusse in questa rassegna rappresentano tuttora argomenti di ricerca sui quali si continua ad investigare anche alla luce di mutate esigenze applicative e dei progressi della tecnologia hardware (ad

[9] Per accelerare l'operazione di fusione le partizioni vengono rappresentate come alberi binari.

esempio memorie temporanee e permanenti sempre più veloci e capaci). Vediamo ora quali sono gli aspetti tuttora in fase di studio.

Uno di questi riguarda il clustering in ambienti *cliente-servente,* dove il servente riceve richieste in modo concorrente e queste devono essere opportunamente gestite.

Inoltre è molto importante la gestione del clustering in presenza di *relazioni multiple* tra gli oggetti. Infatti eseguire il clustering rispetto ad una singola relazione può comportare un degrado delle prestazioni di quelle applicazioni che ne utilizzano altre.

Infine si sta investigando sulla possibilità di ottenere i pesi del *grafo di clustering* modellando i flussi di richieste delle applicazioni in modo da catturare le dipendenze non di una ma di più di due richieste consecutive; questo sposterebbe il problema dal partizionamento di grafi a quello di *ipergrafi.*

Altri campi di ricerca che possiamo elencare sono:

- Combinazione del clustering con tecniche di *buffering intelligente.*
- Supporto da parte del *SGBDOO* di *operazioni orientate agli insiemi,* combinate con tecniche di clustering (anche a livello logico) che consentono di diminuire il numero delle operazioni di I/O in ambiente *cliente-servente.*
- Sviluppo di *modelli di costo* per la misura del raporto *costo/beneficio* delle riorganizzazioni *off-line* (in caso di *clustering statico*) o del *clustering dinamico,* allo scopo di guidare le strategie di *re-clustering.* Questi modelli devono tener conto di molteplici *parametri di controllo.*
- Approfondire le relazioni tra *clustering* e *strategie di controllo della concorrenza.* A volte può essere utile de-clusterizzare parti di un oggetto per aumentare il livello di concorrenza in ambienti *multi-client.*
- Sviluppare *benchmark* per confrontare diverse strategie di clustering utilizzando diversi *parametri di controllo.*

Riferimenti

[Anderson 90]
T. Anderson e al.,
The HyperModel Benchmark,
Proceedings of the EDBT Conference, Venezia, Marzo 1990 in *Lecture Notes in Computer Science*, vol. 416, pp. 317-331 (Berlin: Springer-Verlag).

[Bancilhon 88]
F. Bancilhon e al.,
The design and implementation of O_2, an object oriented database system,
Proceedings of the Second International workshop on object-oriented database systems, edited by K. Dittrich, 1988.

[Bancilhon 92]
F. Bancilhon, C. Delobel, P. Kanellakis,
Building an Object-Oriented Database System: The Story of O_2,
Morgan Kaufmann Publishers, 1992.

[Banerjee 88]
J. Banerjee, W. Kim, S.J. Kim, J.F. Garza,
Clustering a DAG for a CAD database,
IEEE Trans. Software Eng., 14(11):1684-1699, Nov 1988.

[Benzaken 90]
V. Benzaken,
An Evaluation Model for Clustering Strategies in the O_2 Object-Oriented Database System,
3th Int. Conference on Database Theory, Paris, France, December 12-14, 1990,
Proc. In Lecture Note in Computer Science, vol. 470, Springer 1990.

[Bertino 94]
E.Bertino, A.A. Saad, M.A. Ismail,
Clustering techniques in object bases: A survey,
Data and Knowledge Engeneering, 12 (1994), pp. 255-275.

[Breuer 77]
M.A. Breuer,
A class of min-cut placement algorithms,
Proc. of the Design Automation Conference, pp. 284-290, 1977

[Carey 90]
M. Carey e al.,
The Exodus extensible DBMS project: an overview,
Readings in Object-Oriented Database Systems, M. Kauffmann editions, 1990.

[Carey 93]
M.J. Carey, D.J. DeWitt, J.F. Naughton,
The OO7 Benchmark
SIGMOD Record, 22(2): 12-21, 1993.

[Cattel 92]
R. Cattel e J. Skeen,
Object Operation Benchmark
ACM Transactions on Database Syatems, 17(1): 1-31, 1992.

[Chang 89]
E.E. Chang, R.H. Katz,
Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS,
Proc. ACM SIGMOD International Conference on Management of Data, 18(2) pp. 348-357, 1989.

[Chang 90]
E.E. Chang, R.H. Katz,
Inheritance in computer-aided design databases: semantics and implementation issues,
CAD, Vol. 22, No. 8, October 1990

[Cheng 91]
J.R. Cheng, A.R. Hurson,
Effective Clustering of Complex Objects in Object-Oriented Databases,
Proc. ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May, 1991, pp. 22-31.

[Copeland 84]
G. Copeland e D. Maier,
Making Smalltalk Database System,
Proceedings of the ACM SIGMOD, international conference on the management of data, 1984.

[Darmont 95]
J. Darmont, A. Attoui, M. Gougand,
Performance Evaluation for Clustering Alggorithms in Object-Oriented Database Systems,
Database and Expert Systems Applications, 6th International Conference, London, UK, September 4-8, 1995. *Proc. in Lecture Notes in Computer Science*, Vol. 978, Springer 1995, pp 187-96.

[Fiduccia 82]
C. Fiduccia, R. M. Matteyses,
A linear-time heuristic for improving network partitions,
Proc. of the Design Automatic Conference, pp. 175-181, 1982.

[Gerlhof 92]
C. Gerlhof, A. Kemper, C. Kilger, G. Moerkotte,
Clustering in Object Bases,
Technical Report 6/92, University of Karlsruhe, Jun 1992.

[Gerlhof 93]
C. Gerlhof, A. Kemper, C. Kilger, G. Moerkotte,
Partition-Based Clustering in Object Bases: From Theory to Practice,
Foundations of Data Organizzation and Alghoritms, 4th Int. Conference, Chicaco, Illinois, USA, 13-15 October, 1993,
Proceedings in Lecture Notes in Computer Science, Vol. 730, pp 301-316, Springer, 1993.

[He 93]
M. He, A.R. Hurson, L.L. Miller, D. Sheth,
An Efficient storage Protocol for Distributed Object-Oriented Databases,
IEEE Parallel & Distributed Processing, 1993.

[Hornick 87]
M.F. Hornick, S.B. Zdonik,
A shared segmented memory system for an Object-Oriented Database,
ACM TOOIS, 5 (1987) pp. 75-95.

[Hudson 89]
S.E. Hudson, R. King,
Cactis: A self-adaptive, concurrent implementation of an obect oriented database management system,
ACM Trans. Office Inf. Syst., 5(1): pp. 70-95, Jan 1987.

[Joseph 89]
J. Joseph e al.,
Report on the Object-Oriented Database Workshop,
SIGMOD Record, 18(3): pp. 2-11, 1989.

[Kernighan 70]
B. Kernighan, S. Lin,
An efficient heuristic procedure for partitioning graph,
Bell system technical journal, 49(2), pp. 291-307, Feb., 1970.

[Kim 89]
W. Kim, E. Bertino, J. Garza,
Composite Object Revisited,
Proc. ACM SIGMOD Conf., Portlan, OR, June 1989, pp. 337-347.

[Kim 90]
W. Kim, J.F. Garza, N. Ballou, D.Woelk,
Architecture of the ORION Next-Generation Database System,
IEEE Transaction on Knowledge and Data Engeneering, Vol. 2, No. 1, March 1990.

[Kruskal 56]
J.B.Kruskal,
On the shortest spanning subgraphof a graph on the travelling salesman problem,
Proc. of the Amer. Math. Soc., pp. 48-50, 1956.

[Lukes 74]
J. Lukes,
Efficient algorithm for the partitioning of trees,
IBM Journal of Research and Developpement, 18:217-224, 1974.

[Shahookar 91]
K. Shahookar, P Mazumber,
VLSI cell placement techniques,
ACM Computing Surveys, 23(2):143-220, Jun 1991.

[Skarra 86]
A.H. Skarra, S.B. Zdonik, S.P. Reiss,
An object server for an OODB system,
Proc. .1st International Workshop on OODB Systems, 1986, pp. 196-204.

[Stamos 84]
J. Stamos,
Static grouping of samll objects to enhance performance of a paged virtual memory,
ACM Trans. Comp. Syst., 2(2): pp. 155-180, May 1984.

[Tsangaris 91]
M.M. Tsangaris, J.F. Naughton,
A stochastic approach for clustering in object base,
ACM SIGMOD Int. Conf. on Management of Data, Denver, CO, May, 1991, pp. 12-21.

[Tsangaris 92]
M.M. Tsangaris, J.F. Naughton,
On the performance of Object Clustering Techniques,
ACM SIGMOD Int. Conf. on Management of Data, San Diego, CA, June, 1992, pp. 144-153.